Using GPU Acceleration and a Novel Artificial Neural Networks Approach for Ultra-fast Fluorescence Lifetime Imaging Microscopy Analysis

Gang Wu

A Thesis Submitted for the Degree of Doctor of Philosophy

School of Engineering and Informatics

University of Sussex

November 2017

To my beloved parents and wife.

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other University. This dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration, except where specifically indicated in the text.

Signature:

Gang Wu

November 2017

Acknowledgements

PhD study has definitely been an unforgettable and precious journey of my life. It was the first time I ever left my country, and went to a new country with a very different culture. It was a privilege to witness two different cultures. This journey has not only strengthen my research ability, but also influenced my lifestyle, my value and my philosophy.

I would like to thank my supervisors, Dr. David Day-Uei Li and Prof. Thomas Nowotny, for all their invaluable help and support throughout my PhD study. Without their help, I would not be able to get this far. I can always remember how they helped me with my research and papers. They encouraged me to think creatively and work on my own proposal. I would also like to thank all my friends and colleagues in the UK. Especially, Bill Parslow, Janice Britz, and the whole family, which is like my second family.

I thank my parents from the bottom of my heart for their life long support. Also, a special thanks to my wife, who has given me so much happiness. The long geographic distance never separates us, and it has been proved that she is the girl I want to marry with.

Last but not least, I would like to thank the University of Sussex, the China Scholarship Council, and the NVIDIA corporation for their support.

Summary

Fluorescence lifetime imaging microscopy (FLIM) which is capable of visualizing local molecular and physiological parameters in living cells, plays a significant role in biological sciences, chemistry, and medical research. In order to unveil dynamic cellular processes, it is necessary to develop high-speed FLIM technology. Thanks to the development of highly parallel time-to-digital convertor (TDC) arrays, especially when integrated with single-photon avalanche diodes (SPADs), the acquisition rate of high-resolution fluorescence lifetime imaging has been dramatically improved.

On the other hand, these technological advances and advanced data acquisition systems have generated massive data, which significantly increases the difficulty of FLIM analysis. Traditional FLIM systems rely on time-consuming iterative algorithms to retrieve the FLIM parameters. Therefore, lifetime analysis has become a bottleneck for high-speed FLIM applications, let alone real-time or video-rate FLIM systems. Although some simple algorithms have been proposed, most of them are only able to resolve a simple FLIM decay model. On the other hand, existing FLIM systems based on CPU processing do not make use of available parallel acceleration.

In order to tackle the existing problems, my study focused on introducing the state-of-art general purpose graphics processing units (GPUs) to the FLIM analysis, and building a data processing system based on both CPU and GPUs. With a large amount of parallel cores, the GPUs are able to significantly speed up lifetime analysis compared to CPU-only processing. In addition to transform the existing algorithms into GPU computing, I have developed a new high-speed and GPU friendly algorithm based on an artificial neural network (ANN). The proposed GPU-ANN-FLIM method has dramatically improved the efficiency of FLIM analysis, which is at least 1000-folder faster than some traditional algorithms, meaning that it has great potential to fuel current revolutions in high-speed high-resolution FLIM applications.

Contents

Contents vi
List of Figures ix
List of Tables xvi
Nomenclature xvii
Publications and presentations xx
Chapter 1 Introduction 21
1.1 Overview 21
1.2 Background and objectives 21
1.3 Contribution 24
1.4 Structure of the thesis 25
Chapter 2 Fluorescence lifetime imaging microscopy and applications 27
2.1 Overview 27
2.2 Principle of FLIM 27
2.2.1 Fluorescence 27
2.2.2 Fluorescence lifetime and quantum yields 29
2.2.3 Fluorescence lifetime imaging 31
2.3 Time-domain FLIM by TCSPC 33
2.3.1 TCSPC 33
2.3.2 Instrument response function 35
2.3.3 Standard algorithms 36
2.4 FLIM applications 46
2.5 Summary 48
Chapter 3 Graphics processing units and FLIM analysis 49
3.1 Overview 49
3.2 Parallel Computing 49
3.2.1 Brief history of Parallel Computing 50
3.3 Graphics processing units 51
3.3.1 GPUs as programmable parallel processors 51
3.3.2 GPU architecture 55

3.4 CUDA overview 57
3.4.1 CUDA architecture 58
3.4.2 Programming model 59
3.5 GPU applications 68
3.6 GPU acceleration of standard FLIM algorithms 68
3.6.1 Thread-based computing for non-iterative algorithms 69
3.6.2 Block-based computing for iterative algorithms 70
3.7 Optimization of GPU Programming 73
3.7.1 Kernel Configuration 73
3.7.2 Memory Optimization 75
3.7.3 Programming Strategies 78
3.8 Evaluation of standard algorithms 80
3.9 Summary 86
Chapter 4 Artificial neural network and FLIM analysis 87
4.1 Overview 87
4.2 Artificial neural networks 88
4.2.1 A biological neuron 89
4.2.2 Mathematical model of an artificial neuron 90
4.2.3 Feedforward neural network 92
4.2.4 ANN training 93
4.2.5 Applications of ANNs 97
4.3 ANN-FLIM for tail fitting 98
4.3.1 FLIM Model 99
4.3.2 ANN architecture 100
4.3.3 ANN Training 101
4.3.4 Lifetime Calculation 105
4.4 ANN-FLIM including IRF 105
4.4.1 FLIM Model 106
4.4.2 ANN architecture 107
4.4.3 ANN Training 108
4.4.4 Lifetime Calculation 110
4.5 Summary 111
Chapter 5 GPU accelerated ANN-FLIM analysis 112
5.1 Overview 112
5.2 GPU acceleration of ANN-FLIM methods 112
5.2.1 CUDA matrix multiplication 113

5.3 Evaluation of ANN-FLIM algorithms 115
5.3.1 ANN-FLIM for tail-fitting 116
5.3.2 ANN-FLIM for IRF based fitting 121
5.4 ANN-FLIM assessment on experimental data 127
5.4.1 ANN-FLIM for tail-fitting 128
5.4.2 ANN-FLIM for IRF based fitting 130
5.5 Summary 134
Chapter 6 Discussion and Conclusion 135
6.1 Overview 135
6.2 Architecture of GPU accelerated FLIM analysis system 136
6.2.1 Algorithm selection of FLIM analysis 137
6.2.2 Analysis mode 139
6.3 Standard methods 139
6.3.1 Features 139
6.3.2 Limitations 140
6.4 ANN-FLIM methods 141
6.4.1 Features 141
6.4.2 Limitations 142
6.4.3 Further considerations 143
6.5 GPU acceleration 148
6.5.1 Features 148
6.5.2 Limitations 149
6.5.3 Further considerations 150
6.6 Conclusions 150
6.7 Recommendations for future research 152
References 153

List of Figures

Figure 2.1 - A typical form of Jablonski diagram (Modified form [27]). S_0, S_1, and S_2 represent the singlet ground, first, and second excited electronic states respectively. After absorption of a photon, a fluorophore is usually excited to higher excited states, nearly all excited molecules rapidly relax to the lowest excited level of the first excited state S1. After the decay process (which contains non-radiative and radiative decay), the energy level of fluorophore returns to S_0. 28

Figure 2.2 - Principle of FLIM (Modified from [85]). A fluorescent sample is excited with light source, and the emitted photons (photons from light source have been excluded) are captured by a detector. Before a FLIM image is generated, lifetime of each pixel is calculated through a certain analysis method. 32

Figure 2.3 - General principle of TCSPC (Reproduced from [21]). The fluorescence sample is repetitively excited by a pulse light source. During each signal period, the first emitted photon is recorded. The arrival time of each detected photon is measured, and the photon count of the corresponding time bin is incremented by 1. By accumulating all the detected photons, the fluorescence decay histogram is reconstructed. 34

Figure 2.4 - The fitting range of tail-fitting methods, which starts from the peak of the histogram. 37

Figure 2.5 - A single-exponential decay and concept of IEM algorithm (Modified from [42]). 38

Figure 2.6 - The fitting range of IRF based methods. 44

Figure 3.1 - Evolution of CPU and GPU memory bandwidth (Reproduced from [68]). Currently, the memory bandwidth of GPUs is more than 7-fold higher than CPUs'. 52

Figure 3.2 - Evolution of CPU and GPU floating-point performance (Reproduced from [68]). During the past decade, the performance of GPUs has been improved dramatically, which is significantly higher than CPUs performance. 52

Figure 3.3 - Design philosophies of CPUs and GPUs (Reproduced from [68]). Compared with CPU, more transistors are devoted to data processing rather than data caching and flow control for GPU. 53

Figure 3.4 - Diagram of GPU data flow (Reproduced from [69]). Input data is copied from CPU memory to GPU memory through PCIe bus and it is always done by the CPU. 54

Figure 3.5 - Diagram of GPU program execution flow (Reproduced from [69]). CPU is responsible for launching a GPU kernel, then the GPU caches data on chip for performance and executes the program. 54

Figure 3.6 - CPU copies results from GPU memory (Reproduced from [69]). The results of GPU computation are sent back to the CPU memory, and it is also usually done by the CPU. 55

Figure 3.7 - A typical architecture of GPU (Reproduced from [68]). TPC: texture/processor cluster; SM: streaming multiprocessor; SP: streaming processor; ROP: raster operation processor. Each building block contains two SMs (the number of SMs in a building block varies from one generation of GPUs to another). Also, each GPU has its own graphics double data rate (GDDR) DRAM, which is known as global memory. The SMs connect with the DRAMs via an interconnect network. Within a SM, there are a number of SPs that share control logic and instruction cache. Each SM also contains special function units (SFUs), instruction and constant caches, a multithreaded instruction unit, and a shared memory. 57

Figure 3.8 - CUDA architecture (Reproduced from [73]). The general CUDA architecture consists of the following basic ingredients: 1. CUDA parallel compute engines; 2. Operating system kernel-level support; 3. User-mode driver; 4. Instruction Set Architecture named PTX (Parallel Thread Execution) 59

Figure 3.9 - Execution of a CUDA program. First a CUDA program is executed in a CPU. Then a kernel function is launched, and the execution is moved to a GPU by generating a large number of threads. 61

Figure 3.10 - The relationship between threads, blocks, and a grid. The total number of threads is equal to the number of threads per block time the number of blocks. The dimension of a grid is generally dictated by the size of the data being processed or the number of processors of GPUs, which it can greatly exceed. 63

Figure 3.11 - Overview of the CUDA device memory model (Reproduced from [68]). Each thread has access to private local memory and register. Registers are the fastest memory space on a GPU. The local memory is used for data that does not fit into registers. 65

Figure 3.12 - Heterogeneous Programming model (Reproduced from [68]). The CUDA kernels run on the GPU device and the rest of the C program executes on a CPU. 67

Figure 3.13 - The deconstruction of FLIM image. A FLIM image is consist of n×n number of independent pixels. The colour of each pixel is determined by the lifetime of corresponding FLIM histogram. One can learn from this figure that FLIM analysis is highly suitable for parallel computing. In order to speedup FLIM analysis, it is necessary to calculate the lifetime of each pixel. 69

Figure 3.14 - The relationship between Thread, Block and Grid, and GPU implementation of non-iterative algorithms for FLIM analysis (Reproduced from [85]). Each thread processes an entire pixel 70

Figure 3.15 - The relationship between Thread, Block and Grid, and GPU implementation of iterative algorithms for FLIM analysis (Reproduced from [85]). Each thread processes on bin of the histogram of one pixel. 71

Figure 3.16 - The subtraction in GPU and CPU-OpenMP processing (Reproduced from [85]). Subtraction can be finished in only one instruction cycle of one instruction in GPU, whereas there are 64 subtraction cycles in the CPU-OpenMP operation. 72

Figure 3.17 - The summation reduction in GPU and CPU processing (Reproduced from [85]). After only $\log_2 256=8$ steps, the summation reduction result can be acquired in GPU, instead of after 65 steps in the CPU-OpenMP version. 73

Figure 3.18 - GPU-FLIM analysis with mapped pinned memory (Reproduced from [85]). The mapped pinned memory of CPU has been page-locked and mapped for direct access by the GPU. 76

Figure 3.19 - Timeline comparison of sequential and concurrent executions (Reproduced from [85]). The sequential execution has only one stream, however, the concurrent execution has two or more streams. 77

Figure 3.20 - Global memory access: (a) coalesced access; (b) non-coalesced access (Reproduced from [85]). Using coalesced access as much as possible is very important to minimize the necessary bandwidth. 78

Figure 3.21 - Example of avoiding warp divergence (Reproduced from [85]). In some circumstances, shifting elements by changing the index can be used to reduce the divergence. 79

Figure 3.22 - Example of maximizing hardware usage (Reproduced from [85]). In this case, two groups of operations are running simultaneously, and the number of idle threads is reduced. 79

Figure 3.23 - FLIM images of single-exponential decays: (a) theoretical lifetime image; (b) IEM; (c) CMM; (d) LSM (Reproduced from [85]). The theoretical FLIM image is a square with 4 color bars. All algorithms generate effective results in agreement with the known ground truth, although IEM is not as good as others. The averaged relative precision values (the average value of σ_τ/τ) of IEM, CMM, and LSM are 0.15, 0.026, and 0.03. 81

Figure 3.24 - Images based on synthesized bi-exponential data: (a) theoretical average lifetime image; (b) PM; (c) BCMM; (d) GA-32 (Reproduced from [85]). All the algorithms generate satisfactory results when $f_D > 0.5$, whereas the GA shows the great potential as it can resolve a wider range of f_D. The averaged relative precision values of PM, BCMM, and GA are 0.048, 0.046, and 0.013. 82

Figure 3.25 - Acquisition time and GPU image analysis time for IEM with different image resolutions (Reproduced from [85]). Compared to the acquisition time, the analysis time is negligible. 84

Figure 3.26 - Acquisition time and GPU image analysis time for LSM with different image resolutions (Reproduced from [85]). For LSM, the analysis time is also less than the acquisition time. 85

Figure 3.27 - Acquisition time and GPU image analysis time for GA with different image resolutions (Reproduced from [85]). Due to the complexity of GA, the analysis time is around 2-fold longer than the acquisition time. 85

Figure 3.28 - The ratio of GPU image analysis time and acquisition time for three FLIM algorithms with different image resolutions. For IEM and other simple algorithms, the ratio is rather stable across image resolutions. Since the acquisition time falls proportionally to the number of pixels, the analysis time also is in proportion to the

number of pixels in a consistent manner. However, as for the iterative algorithms, the ratio grows when the image size decreases, and this means the analysis times are no longer in proportion to the number of pixels. The more complicated the algorithm is the faster the ratio grows. 86

Figure 4.1 - Structure of a biological neuron. The biological neurons consist of four basic components: a cell body (soma), dendrites, an axon, and synapses. 89

Figure 4.2 - A basic artificial neuron. Each neuron has several inputs and one output. It consists of weights summation (determines how the network inputs are combined inside a neuron) and activation function (determines the output). 91

Figure 4.3 - Neuron transfer functions: (a) linear, (b) ReLU, (c) log-sigmoid, (d) tan-sigmoid. 91

Figure 4.4 - A multilayer perceptron (MLP) structure, which contains one input layer, one hidden layer, and one output layer. 93

Figure 4.5 - A diagram of supervised learning. The learning algorithm attempts to improve the mapping implied by data through updating the weights of the network. ... 95

Figure 4.6 - The error surface (the relationship between weights parameters and error). Especially, the red points are the local minima. The objective of training process is to find the best weights related to the global minima, so that the ANN is able to imitate the target system as much as possible. However, in practice, it is not possible to generate such an error plot or find the best weights by observing, since the number of weights is very large in most cases. It is necessary to design an algorithm that can find the global or local minima. 95

Figure 4.7 - Principle of the ANN-FLIM analysis: Selected photon counts of each pixel are used as the rate inputs to an ANN that approximates the unknown function from the high-dimensional space of photon count histograms to decay constants and other parameters of the lifetime function, from which FLIM images can be generated. 99

Figure 4.8 - The architecture of the ANN (Modified from [116]). The ANN contains 2 hidden layers (with sigmoid transfer function based neurons), one input layer and one linear output layer. To be more specific, each layer has 64, 32, 128, 4 neurons, respectively. 101

Figure 4.9 - Preparation of the training samples (Reproduced from [116]). First, a set of α vectors from within the working range are chosen. Next, the theoretical fluorescence histogram is generated based on α by following the decay model. Then the TCSPC decay signals can be simulated by combining Poisson noise with the theoretical histogram values. The noisy TCSPC decays are fed to the maximum likelihood estimation, and the results of the MLE are used as the training targets of the ANN. 103

Figure 4.10 - ANN-FLIM lifetime calculation procedure. The ANN-FLIM mapping contains tail selecting, pre-processing, neural network computation, and post-processing. 105

Figure 4.11 - Principle of the ANN-FLIM analysis: Photon counts of each pixel are used as the rate inputs to an ANN that approximates the unknown function from the high-

dimensional space of photon count histograms to decay constants and other parameters of the lifetime function, before FLIM images can be generated. 106

Figure 4.12 - The architecture of the ANN. The ANN contains 4 hidden layers (with sigmoid transfer function based neurons), one input layer and one linear output layer. To be more specific, each layer has 128, 32, 32, 64, 64, 4 neurons, respectively. 108

Figure 4.13 - Preparation of the training samples. First, the IRF of the target FLIM equipment is acquired. Then, a set of vectors $\alpha = [K, f_D, \tau_F, \tau_D]$ is chosen to generate FLIM decays for training. Next, the intrinsic decay function $I(t)$ is generated based on α. Then the TCSPC decay signals can be simulated by combining Poisson noise with the theoretical histogram values, resulting in the final histogram as illustrated by the red curve in the middle of figure. The noisy TCSPC decays are fed to the maximum likelihood estimation (MLE) block, with α as the initial value, and the results ($\alpha^* = [K^*, f^*_D, \tau^*_F, \tau^*_D]$) of the MLE are used as the training targets of the ANN. Finally, the photon counts in the first few bins before the rising part of the histogram are neglected them in order to increase processing speed. 109

Figure 4.14 - ANN-FLIM lifetime calculation procedure. The ANN-FLIM mapping contains pre-processing, neural network computation, and post-processing. 110

Figure 5.1 - The computation flow of neural networks on GPU. Each layer contains a matrix multiplication and a non-linear operation, and the results of the network is calculated layer by layer. 113

Figure 5.2 - A CUDA implementation of matrix multiplication with shared memory (Reproduced from [68]). J_{sub} is the output matrix of two rectangular sub matrices of P and W. In order to optimize the CUDA program and fit into the device's resources, these two rectangular matrices are divided into as many square matrices of dimension block_size as necessary. As a result, J_{sub} is acquired as the sum of the products of divided square matrices. 115

Figure 5.3 - Performances of τ_F calculated by ANN and LSM: (a) and (b) F-value; (c) and (d) the bias; (e) areas where ANN has better precision performance (green areas); (f) areas where ANN has better accuracy performance (green areas). For both precision and accuracy performance, the smaller result means the better performance. From these figures, one can learn that the optimized regions (for the F-value and the bias) of LSM are different from those of ANN. Figures 5.3 (a), (b) and (e) show that ANN can provide a wider optimized area for the F-value for all τ_F. Figures 5.3(c), (d) and (f) show that LSM produces slightly less biased τ_F. Although there are differences between them, their estimations are in the same order. 118

Figure 5.4 - Performances of f_D calculated by ANN and LSM: (a) and (b) F-value; (c) and (d) the bias; (e) areas where ANN has better precision performance (green areas); (f) areas where ANN has better accuracy performance (green areas). For both precision and accuracy performance, the smaller result means the better performance. From these figures, one can learn that the optimized regions (for the F-value and the bias) of LSM are different from those of ANN. Figures 5.4 (e) and (f) show that ANN can provide a wider optimized area for both F-value and bias for f_D. 119

Figure 5.5 - Performances of τ_D calculated by ANN and LSM: (a) and (b) F-value; (c) and (d) the bias; (e) areas where ANN has better precision performance (green areas); (f) areas

where ANN has better accuracy performance (green areas). For both precision and accuracy performance, the smaller result means the better performance. Figures 5.5(a), (b) and (e) show that ANN can provide better precision performance for all f_D. Figures 5.5(c), (d) and (f) show that ANN also produces less biased f_D.....120

Figure 5.6 - Simulated IRF based on measured data. The green lines demarcate the full width of half-maximum, which spans time bin 39 to 50.....122

Figure 5.7 - ANN and LSIR performance comparison (500 counts): F-value of (a) $f_{D,}$, (c) τ_F, and (e) τ_D; the bias of (b)) $f_{D,}$, (d) τ_F, and (f) τ_D. The results of ANN are given in red curves, and it is obvious that the ANN can provide comparable or even better outcomes (except when N_C is low and f_D is small). Although, when f_D is close to 0, the bias of f_D generated by the ANN is higher than the LSIR, the ANN is able to provide similar results when f_D is getting larger. As for the bias of τ_F, or τ_D, except when f_D is very small, both methods can provide satisfactory outputs. Moreover, it is clear that when these two methods have the similar bias performance, the performance of ANN-FLIM in terms of precision is better for ANN-FLIM**.** 124

Figure 5.8 - ANN and LSIR performance comparison (1500 counts): F-value of (a) $f_{D,}$, (c) τ_F, and (e) τ_D; the bias of (b)) $f_{D,}$, (d) τ_F, and (f) τ_D. ANN has potential to outperform LSIR. Besides its superior precision performance over LSIR, the bias of f_D is significantly improved when the photon counts increase. 125

Figure 5.9 - ANN and LSIR performance comparison (2500 counts): F-value of (a) $f_{D,}$, (c) τ_F, and (e) τ_D; the bias of (b)) $f_{D,}$, (d) τ_F, and (f) τ_D. ANN has potential to outperform LSIR. Besides its superior precision performance over LSIR, the bias of f_D is significantly improved when the photon counts increase. 126

Figure 5.10 - (a) Intensity image, (b) ANN and (c) LSM τ_F images, (d) ANN and (e) LSM merged intensity and $\tau_{Average}$ images (Reproduced from [116]). ANN is capable of extracting the features of the sample, and it shows similar results to those of LSM, especially for the images of average lifetime.....129

Figure 5.11 - (a) Lifetime and (b) f_D histograms of the experimental data (Reproduced from [116]). ANN produces similar τ_F , $\tau_{Average}$ and f_D histograms with LSM, except that LSM has more invalid pixels around $\tau_F \sim 0$ns and there is a slight difference in τ_D.....129

Figure 5.12 - Background noise calibration of the experimental IRF.....131

Figure 5.13 - (a) Intensity image, (b) ANN and (c) LSIR $\tau_{Average}$ images combined with intensity. (d) ANN and (e) LSIR τ_D images. Two algorithms generate similar merged FLIM images. Whilst, the lifetime image generated by LSIR is noisier. However, due to different bias performances, these two methods produce different τ_D images.....132

Figure 5.14 - (a) Lifetime and (b) f_D histograms of the experimental data. ANN-FLIM can have similar performances as LSIR, except the bias of τ_D, and the results generated by ANN-FLIM have higher precision.....132

Figure 5.15 - (a) New ANN τ_D image, (b) LSIR τ_D image. Although the bias of τ_D between these two algorithms is unneglectable, lifetime images with similar contrast can be acquired when the overall bias is eliminated. 133

Figure 6.1 - The architecture of GPU accelerated FLIM analysis system. First, we have to decide the model of exponential decay and select appropriate algorithm for lifetime analysis. After finishing system setup, FLIM data will be transferred to GPU devices for lifetime calculations. Finally, image generator is able to generate FLIM images based on the results from GPU. In addition, if users are not sure which algorithm works best, they can select different algorithms and determine the final algorithm by observing the FLIM images. 136

Figure 6.2 - A simple diagram of algorithm selection for FLIM analysis. First, to decide which fitting model to follow: tail-fitting or IRF based fitting. Then, final algorithm under selected model can be determined based on the features of each algorithm. 137

Figure 6.3 - Architectures of networks (a) NET1: two hidden layers (32, 64 neurons), (b) NET2: two hidden layers (32, 128 neurons), (c) NET3: four hidden layers (16, 16, 64, 64 neurons). More specifically, NET1 has less total neurons (96) in hidden layers, and NET2 and NET3 have the same number of total neurons (160) in hidden layers. Although NET2 and NET3 have the same number of neurons, NET3 has much less weight variables (3392) than NET2 (6272). Moreover, as demonstrated on the figures, last hidden layer (or last two hidden layers) are not fully connected. This is based on the knowledge that not all parameters are need for FLIM analysis, and this configuration can speedup lifetime calculation by eliminating unnecessary neurons. 145

List of Tables

Table 3.1 CUDA variable type qualifier. 66
Table 3.2 Operation time under different block size of LSM-GPU (Reproduced from [85]). 75
Table 3.3 Operation time under different grid size of LSM-GPU (Reproduced from [85]). 75
Table 3.4 Comparison between coalesced access and non-coalesced access of global memory. 78
Table 3.5 Details of bi-exponential simulations 82
Table 3.6 Processing time for single-exponential decay (unknown: τ, K) (Reproduced from [85]) 83
Table 3.7 Processing time for double-exponential decay (unknown: τ_F, K, f_D) (Reproduced from [85]) 84
Table 3.8 Results from CUDA profiler for each algorithm (Reproduced from [85]) 84
Table 4.1 Comparison of ANNs trained with different samples 102
Table 5.1 Simulation setup of ANN-FLIM 116
Table 5.2 CPU processing time of ANN and LSM for synthesized data 121
Table 5.3 GPU acceleration of ANN and LSM for synthesized data 121
Table 5.4 Simulation setup of ANN-FLIM 122
Table 5.5 Processing time of ANN, LSIR and LSD-LE for synthesized data 127
Table 5.6 GPU acceleration of ANN, LSIR and LSD-LE for synthesized data 127
Table 5.7 CPU processing time of ANN and LSM for experimental data 130
Table 5.8 GPU processing time of ANN and LSM for experimental data 130
Table 5.9 CPU processing time of ANN and LSM for experimental data 133
Table 5.10 GPU processing time of ANN and LSM for experimental data 133
Table 6.1 Feature of FLIM algorithms. 138
Table 6.2 Performance of networks with different architectures and training configurations 146

Nomenclature

GPUs	Graphics processing units
ANN	Artificial neural network
FLIM	Fluorescence lifetime imaging microscopy
FRET	Förster resonance energy transfer, or fluorescence resonance energy transfer
TD	Time-domain
FD	Frequency-domain
TGSPC	Time-gated single photon counting
TCSPC	Time-correlated single-photon counting
PMT	Photomultiplier tube
SPAD	Single-photon avalanche diode
IRF	Instrument response function
S_0	Singlet ground state
S_1	First excited electronic state
S_2	Second excited electronic state
$h\nu_A$	The energy that has been absorbed by fluorophore
Q	Quantum yield
Γ	The emissive rate of the fluorophore
k_{nr}	The non-radiative decay rate
τ	Fluorescence lifetime

TDC	Time-to-digital convertor
MLE	Maximum likelihood estimation
LSM	Least square method
GA	Global Analysis
IEM	Integral equation method
CMM	Center of mass method
PM	Phasor method
BCMM	Bi-decay CMM
LSIR	Nonlinear least squares iterative reconvolution method
LSD-LE	Laguerre expansion based least squares deconvolution method
τ_F	Lifetime of receptor
τ_D	Lifetime of donor
f_D	Proportion of protein interaction
K	Pre-scalar
CM	Center of mass
ω	Laser repetition angular frequency
$y(t)$	Fluorescence decay histogram
CUDA	Compute Unified Device Architecture
MPPs	Massively parallel processors
SMs	Streaming multiprocessors
SPs	Streaming processors
SIMT	Single-instruction, multiple-thread
CGMA	Compute to global memory access
MLP	Multilayer perceptron

BP	Backpropagation training algorithm
GD	Gradient descent
DNNs	Deep neural networks
FWHM	Full width at half maximum

Publications and presentations

1. Wu, G., Nowotny, T., Chen, Y., & Li, D. D. U. GPU acceleration of time-domain fluorescence lifetime imaging. Journal of biomedical optics, 21(1), 017001-017001, 2016.
2. Wu, G., Nowotny, T., Zhang, Y., Yu, H. Q., & Li, D. D. U. Artificial neural network approaches for fluorescence lifetime imaging techniques. Optics letters, 41(11), 2561-2564, 2016.
3. Zhang, Y., Cuyt, A., Lee, W. S., Bianco, G. L., Wu, G., Chen, Y., & Li, D. D. U. Towards unsupervised fluorescence lifetime imaging using low dimensional variable projection. Optics Express, 24(23), 26777-26791, 2016.
4. Wu, G., Nowotny, T., & Li, D. D. U. GPU enhanced fluorescence lifetime imaging processors with non-iterative and iterative algorithms. Microscience Microscopy Congress, MMC2014, Manchester, UK, 2014.
5. Wu, G., Nowotny, T., Zhang, Y., Yu, H., & Li, D. D. U. Artificial neural network approaches for time-domain FLIM imaging analysis. FIB2016, London, UK, 2016.
6. Wu, G., Nowotny, T., Chen, Y., & Li, D. D. U. GPU Acceleration of non-iterative and iterative algorithms in Fluorescence Lifetime Imaging Microscopy. GTC2016, San Jose, USA, 2016.
7. Wu, G., Nowotny, T., & Li, D. D. U. High-speed fluorescence lifetime imaging techniques based on artificial neural networks. GTC2017, San Jose, USA, 2017.

Chapter 1 Introduction

1.1 Overview

This research harnesses general purpose Graphics Processing Units (GPUs) and artificial neural network (ANN) techniques aiming to address the computational complexity in designing a high-speed fluorescence lifetime imaging microscopy (FLIM) analysis system for biological sciences. This system is able to offer a much faster analysing speed compared with the existing systems, and has great potential to fuel current revolutions in biological research. This chapter provides an introduction to the research performed, by introducing the research background and objectives. At the end of the chapter, the structure and organisation of this thesis are outlined.

1.2 Background and objectives

Cancer, which affects all of humankind, is a leading cause of disease worldwide. Results from GLOBOCAN indicated that there were an estimated 14.1 million new cases diagnosed in 2012 with 8.2 million estimated deaths [1]. Early detection of cancer significantly improves prognosis and offers the best hope for a cure.

Fluorescence lifetime imaging microscopy (FLIM) technologies, which are capable of visualising local molecular and physiological parameters in living cells [2-4], have been widely used in the biological sciences. FLIM is a technique for imaging the differences in the excited state fluorescence decay rate from a fluorescent sample. It provides a powerful tool to detect, visualize, and investigate the structure and function of biological systems, by measuring the lifetimes of individual fluorophores with microscopic spatial resolution [5]. During the past few decades, having benefited from advances in a variety

of technologies, including probe chemistry, confocal laser scanning, multiphoton excitation, photon detecting technology, computers, and genetically expressed fluorophores such as GFP, the use of fluorescence microscopy has grown significantly [5, 6]. In addition to molecular environment monitoring (e.g., pH, O_2, Ca^{2+}, ion concentrations, and temperature), clinical chemistry, and DNA sequencing, FLIM has also been used for local viscosity (known as 'molecular rotors'), cell identification, and to identify and localise nanoparticles [5, 7]. In contrast to intensity imaging, fluorescence lifetime is insensitive to artifacts such as sample thickness, variations in excitation intensity, probe concentrations and optical loss [5]. Instead, fluorescence lifetime is known to be sensitive to the local environments of fluorophores, and therefore can be used to reveal information such as pH, O_2, Ca^{2+}, temperature, and molecular associations [6].

Förster resonance energy transfer (FRET) [8], also known as fluorescence resonance energy transfer (FRET), is a mechanism describing an interaction between two fluorescent molecules within several nanometres. It involves the nonradiative excitation energy transfer from an excited molecule, the donar, to a second, ground-state molecule, the acceptor, in its direct vicinity [9]. FRET processes are driven by long-range dipole–dipole coupling interactions [10]. The efficiency of FRET is inversely proportional to the sixth power of the separation distance between the donor and acceptor. It is extremely sensitive to small changes in distance (in the range of 2-10 nm), so it is used as a molecular ruler. Due to the energy transfer, both the fluorescence intensity and the fluorescence lifetimes of the two molecules change. FRET is usually used to investigate protein interactions. With the help of FRET techniques, FLIM-FRET has been used to investigate protein-protein interactions in live cultured cells, e.g., for cancer diagnosis [11]. In addition, FLIM technologies have also been used for brain tumor image-guided surgery [12], cancer diagnosis [11], cancer therapies [6, 7], drug developments [13], and clearance in tumors [14].

After absorption of a photon, a fluorophore, which is usually excited to higher energy states, will drop to the ground state. The fluorescence lifetime is the average time that a fluorophore remains in an excited state prior to returning to the ground state [5].

Therefore, FLIM is able to produce spatially resolved images of fluorescence lifetime, providing additional dimension of information for visualizing fluorophores and an extra source of contrast [5]. FLIM measurements can be implemented in two different domains according to the methods of data acquisition: time-domain (TD) and frequency-domain (FD). Both TD and FD measurements are able to recover the lifetime parameters with comparable temporal resolution and discrimination. TD measurements are more intuitive implementations that directly reconstruct the exponential decay of fluorescence following a temporal pulse of excitation light [15]. On the other hand, for FD measurements, a continuous sinusoidally modulated light source is used for excitation [5]. The choice of FLIM implementation depends on the particular application: FD systems are better for evaluating short-lifetime measurements, whereas TD systems are more flexible for large temporal ranges, and especially for long-lifetime measurements [5, 15]. Recent advances in image sensors and microscopy technologies have radically boosted TD FLIM acquisition, and this research focuses on TD FLIM.

Time-domain techniques include time-gated single photon counting (TGSPC) [16-18] and time-correlated single-photon counting (TCSPC) [19, 20] methods. Due to superior temporal resolution and recent developments in advanced laser sources and cameras, TCSPC techniques have become the gold standard solutions for TD FLIM experiments [7, 21, 22]. TCSPC techniques are used to measure the time between an excitation laser pulse and the arrival of the emitted fluorescence photon at the photon detector. TCSPC has very high accuracy of time measurement and a high efficiency of detecting fluorescence photons. Traditional TCSPC measurements employ one photomultiplier tube (PMT) as the photon detector, and they can have multiple channels to boost the acquisition. Recent development of CMOS technologies which have allowed single-photon detectors to be fabricated in 2D arrays in low-cost silicon process, have dramatically improved FLIM measurements. More specifically, highly parallel single-photon avalanche diode (SPAD) arrays, which are similar to a multi-PMT TCSPC system, have significantly enhanced the parallelism of data acquisition [23, 24]. With latest multifocal multiphoton FLIM techniques, the acquisition rate of high resolution FLIM has been dramatically improved [22]. FLIM acquisition rate has grown from a frame

every few minutes to several frames per second [25], making FLIM promising for live cell imaging.

These fast imaging systems, however, generate massive data throughput posing greater challenges to image analysis. For example, a 3D live cell FLIM experiment can easily generate tens to hundreds of images resulting in an image dataset of over 10GBytes in size per experiment. What's more, the analysis speed of traditional iterative curve-fitting software tools have not improved at the same rate. The lack of faster analysis for these new FLIM instruments is restricting the ability of cell biologists to study fast cellular processes.

The overall scope of this research was to introduce the latest general purpose graphics processing units (GPUs) and artificial neural network (ANN) techniques to build a comprehensive high-speed, high-precision FLIM analysis tool based on existing FLIM analysis systems, and to fuel current revolutions in biological research. In more detail, the main objectives of this study are to:

a) utilise many-core GPUs to accelerate existing FLIM systems,
b) design high-efficiency and GPU friendly FLIM algorithms in order to build a real-time FLIM analysis system,
c) build a comprehensive GPU accelerated FLIM analysis system combining standard algorithms and new algorithms, to be sufficient for all FLIM applications.

1.3 Contribution

The key contribution of this thesis is to design a high-speed FLIM analysis tool. To be more specific, by transferring data analysis onto the many-core GPUs and by introducing ANN techniques, the following features have been made possible:

a) I have implemented most of the commonly used existing FLIM algorithms on GPUs, and allowed FLIM algorithms to fully utilise a large number of parallel processors. Compared with CPU-OpenMP (parallel computing with multiple

CPU cores) based analysis, GPU based FLIM analysis has significantly sped up lifetime calculations.

b) A novel high-speed FLIM analysis method based on ANN has been proposed, and it is the first time that ANN has been successfully introduced in FLIM analysis. Without iterative searching procedures or initial procedures, the proposed ANN-FLIM method can generate lifetime images at least 180-fold faster than conventional least squares curve-fitting software tools.

c) A more advanced ANN-FLIM method, considering the instrument response function (IRF) of the FLIM instrument, has also been introduced. This method is highly parallelizable, which can further exploit the capability of GPUs, and the GPU based ANN-FLIM method has made Real-time FLIM analysis achievable.

d) A high-speed FLIM analysis system accelerated by GPUs has been introduced, which provides effective analysis solutions for most applications. For FLIM analysis, there is no single algorithm that can replace others, so it is necessary to choose the right algorithms for each application's specific requirements. This system provides a comprehensive solution that contains most of the standard FLIM methods and newly proposed ANN-FLIM methods, and users can easily switch between FLIM methods.

1.4 Structure of the thesis

This thesis is organised into seven chapters. Chapter 2 introduces the basic concepts and applications of FLIM techniques. In particular, this chapter presents the way in which fluorescence lifetimes are detected, and how FLIM techniques are used for biological sciences research. Additionally, commonly used analysis algorithms for current FLIM systems are described.

A study of high-performance parallel computing and GPUs is presented in Chapter 3. Different from CPUs, GPUs have highly parallel structure which are more efficient for algorithms where the processing of large blocks of data is done in parallel. All FLIM algorithms are suitable for GPU acceleration, and GPUs can significantly speed up

lifetime calculations compared to CPU-only analysis. In addition, existing FLIM algorithms have been implemented on GPUs including some important optimization suggestions. Moreover, the performance of GPU accelerated standard FLIM algorithms have been evaluated with synthesized data.

Chapter 4 presents a new FLIM analysis method based on ANN. This chapter begins with the basic concepts of ANN. The architecture of ANN has been customized for this specific application, which allows it to perform a much faster lifetime analysis. Also, the preparation of the ANN-FLIM method has been demonstrated, including the ANN training and lifetime calculation.

In Chapter 5, the implementation of GPU acceleration for ANN-FLIM methods are illustrated. Results are presented regarding the achieved precision performance and acceleration of the GPU accelerated ANN-FLIM methods. This was evaluated by carrying out a Monte Carlo simulation which was compared with the results obtained from using the traditional analysis tool. Furthermore, to demonstrate the applicability of the system, experimental data has also been used.

Finally, Chapter 6 summarises the results and findings of the thesis, draws the conclusions of this research, and presents an outlook for future developments.

Chapter 2 Fluorescence lifetime imaging microscopy and applications

2.1 Overview

FLIM is an advanced imaging technique that has been widely used in biological sciences. In addition to locating fluorescent molecules (fluorophores), it can also generate images based on the temporal decay rate of the fluorescence emitted from fluorophores [5]. In contrast to traditional intensity imaging, FLIM is sensitive to the local environment of fluorophores , but insensitive to intensities of light sources and probe concentrations [5]. Due to this advantage, FLIM technologies are often the most reliable way to measure FRET, and used for studying complex cellular biological events. For time-domain (TD) FLIM systems, due to superior temporal resolution and recent developments in CMOS technology, time-correlated single-photon counting (TCSPC) methods have become the gold standard for FLIM [7, 21, 22]. This chapter first introduces the basic concepts of fluorescence lifetime, and then discusses standard FLIM algorithms for TD FLIM. The applications of FLIM are then presented.

2.2 Principle of FLIM

2.2.1 Fluorescence

In the biological sciences, the use of fluorescence has experienced a significant growth during the past few decades [6]. The use of fluorescence has also been dramatically expanded to biotechnology, environmental monitoring, medical diagnostics, DNA sequencing, forensics, and genetic analysis [5, 6].

The first optical phenomenon of fluorescence was reported by Sir John Frederick William Herschel in 1845, where he described it as “extremely vivid and beautiful celestial blue colour” [26]. As one category of luminescence, fluorescence is the emission of light from an electronically-excited substance that has absorbed light or other electromagnetic radiation [6].

A typical form of Jablonski diagram [27], Figure 2.1 schematically illustrates the processes that occur between the light absorption and emission. In this figure, the singlet ground, first, and second excited electronic states are represented by S_0, S_1, and S_2 respectively, and the horizontal lines illustrate different excited energy levels of the fluorophore.

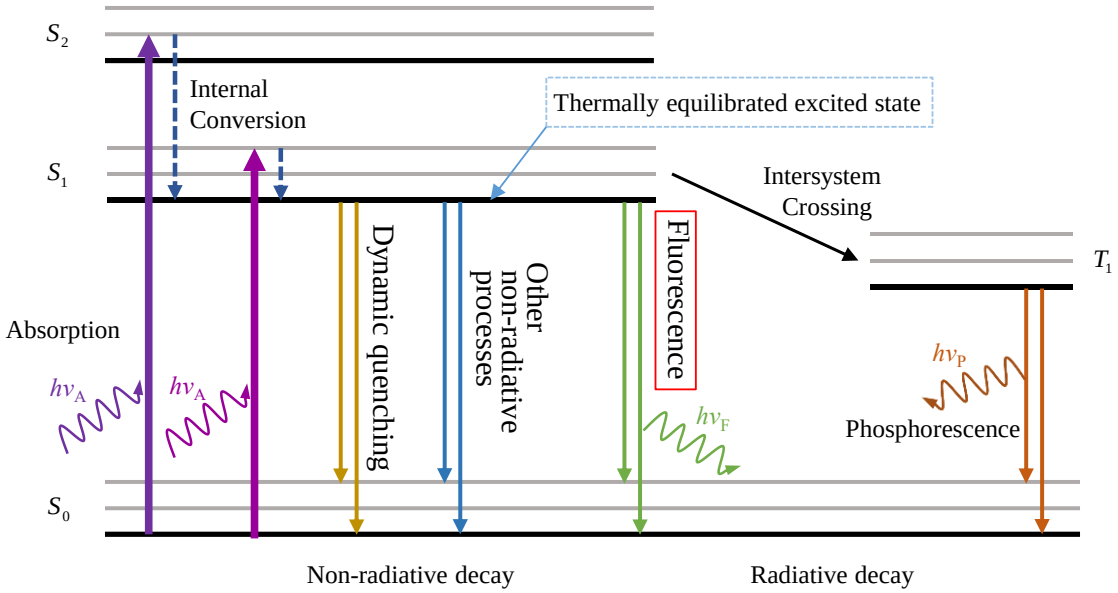

Figure 2.1 - A typical form of Jablonski diagram (Modified form [27]). S_0, S_1, and S_2 represent the singlet ground, first, and second excited electronic states respectively. After absorption of a photon, a fluorophore is usually excited to higher excited states, nearly all excited molecules rapidly relax to the lowest excited level of the first excited state S1. After the decay process (which contains non-radiative and radiative decay), the energy level of fluorophore returns to S_0.

After absorption of a photon, a fluorophore is usually excited to higher excited states (either S_1 or S_2, as illustrated in the Figure 2.1). This process can happen only when the

fluorophore has absorbed greater energy than the difference between the higher excited states and singlet ground level, as demonstrated in Eq. (2.1).

$$h\nu_A \geq \Delta E_{S_0 \rightarrow S_1(S_2)} \tag{2.1}$$

where $h\nu_A$ represents the energy that has been absorbed by fluorophore ($h\nu = hc/\lambda$, with λ the wavelength, c the speed of light and h Planck's constant), and $\Delta E_{S0 \rightarrow S1(S2)}$ is the energy difference between S_0 and S_1 or S_2.

In condensed phases, nearly all excited molecules rapidly relax to the lowest excited level of the first excited state S_1, which is known as internal conversion. Since this process (occurs within 10^{-12}s) and is usually complete prior to emission, the decay process (which contains non-radiative and radiative decay) generally starts from a thermally equilibrated excited state, as indicated in the Figure 2.1. The electronic properties of an isolated fluorophore decide the radiative decay rate. The non-radiative decay, in which the excitation energy is usually dissipated as heat, includes a number of molecular interactions, such as quenching, energy transfer, solvent interactions and FRET, while radiative decay is responsible for fluorescence emission, where detectable photons are generated. Due to the existence of the non-radiative decay (e.g., thermal relaxation), the energy of the photon from fluorescence emission ($h\nu_F$) is always lower than the energy of the absorbed photon ($h\nu_A$).

In addition, another conversion called intersystem crossing can also occur, where molecules in the S_1 state can undergo a spin conversion to the first triplet state T_1 as illustrated in the Figure 2.1. The radiative decay from T_1 to S_0 state is called phosphorescence, which is separate to the process of fluorescence.

2.2.2 Fluorescence lifetime and quantum yields

A fluorophore is a fluorescent chemical compound capable of re-emitting light excitation through the process of fluorescence. Two of the most important characteristics of a fluorophore are the quantum yield and fluorescence lifetime.

Quantum yield is the efficiency of the energy transferred from excitation light to emitted fluorescence, i.e., the number of emitted photons relative to the number of absorbed photons. It is determined by the radiative and non-radiative decays of the fluorophore, as in Eq. (2.2) [6].

$$Q = \frac{\Gamma}{\Gamma + k_{\mathrm{nr}}} \tag{2.2}$$

where Γ is the emissive rate of the fluorophore, and k_{nr} is the non-radiative decay rate which has included all possible processes.

One can see that $0 \leq Q \leq 1$, and the quantum yield can be close to unity when the radiative decay rate is much higher than the radiationless decay rate, i.e., $\Gamma >> k_{\mathrm{nr}}$. Usually, higher Q leads to higher fluorescence efficiency, and substances with the largest quantum yields display the brightest emissions. In fluorescence applications higher Q is more favourable. However, due to the existence of the Stokes shift [28], the energy yield of fluorescence is always less than unity.

The fluorescence lifetime is defined by the average duration (near 10ns) the molecule remains in the excited state prior to return to the ground state [5], which is given in Eq. (2.3). In fact, fluorescence emission is a random process, where few photons can be precisely emitted at the time τ.

$$\tau = \frac{1}{\Gamma + k_{\mathrm{nr}}} \tag{2.3}$$

Fluorescence lifetimes are dependent on the intrinsic properties of the fluorophore, but are insensitive to the factors affecting intensity. There are a variety of intensity based artifacts which might be introduced during the measurement, such as variation in excitation source intensity, detection gain setting, variation in sample fluorophore concentration, photobleaching, and microscope focusing [5]. Moreover, due to the fact that lifetime is also influenced by the non-radiative decay, fluorescence lifetimes are also dependent on the microenvironment of fluorophore, such as pH, O2, Ca2+, Cl-, polarity,

ion concentration, temperature, and relaxation through collisional (dynamic) quenching and FRET [5, 29].

2.2.3 Fluorescence lifetime imaging

Fluorescence measurements are broadly classified into two categories, namely, steady-state (static) and time-resolved (dynamic) [6, 30]. Steady-state measurements involve constant illumination of the sample with a continuous beam of light, and observation of the intensity or emission spectrum. Because the time scale of fluorescence is in fractions of a nanosecond, most of the measurements are steady state measurements.

On the other hand, the time-resolved measurement is used for measuring intensity decay or anisotropy decay with high-speed detection systems [6]. In this measurement, the sample (a population of fluorophores) is excited by a pulse of light, where the pulse has a shorter width than the decay time of the sample. In fact, considering that time-resolved measurement provides data on intensity decay, a steady-state observation is the average of the time-resolved phenomena. For example, consider a fluorophore that displays a mono-exponential fluorescence decay, the intensity profile is given by

$$I(t) = I_0 e^{-t/\tau} \tag{2.4}$$

where I_0 is the intensity at t=0 (immediately following the excitation pulse), and τ is the single decay time.

Time-resolved fluorescence measurements are superior to steady-state measurements, although sophisticated and expensive instrumentation is required. They are capable of providing much of the molecular information that steady-state measurements cannot give, about the dynamical processes which range from fractions of a nanosecond to picoseconds.

FLIM, which belongs to the class of time-resolved fluorescence measurements, is an imaging technique to create spatially resolved images of excited state lifetimes in a fluorescent microscopic sample. The principle of FLIM is demonstrated in Figure 2.2,

where for all FLIM systems a fluorescent dye is activated by the intensity-modulated or pulsed excitation light, and the fluorescence is measure time-resolved.

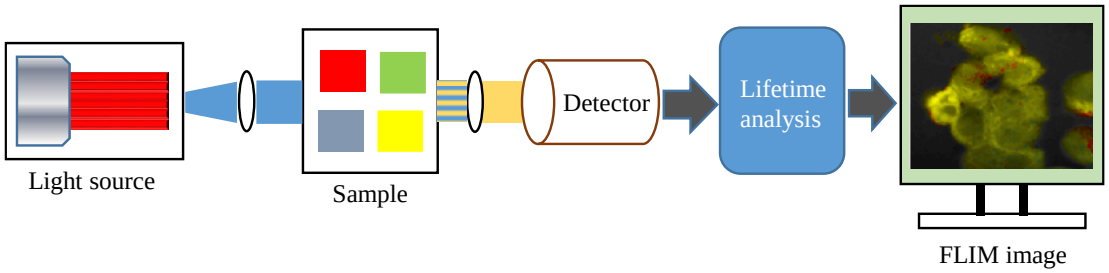

Figure 2.2 - Principle of FLIM (Modified from [85]). A fluorescent sample is excited with light source, and the emitted photons (photons from light source have been excluded) are captured by a detector. Before a FLIM image is generated, lifetime of each pixel is calculated through a certain analysis method.

According to the methods of excitation and data acquisition, FLIM systems can be classified into time-domain (TD) and frequency-domain (FD) techniques. The objective of both TD and FD techniques is to obtain the lifetime, and they are both able to provide comparable temporal resolution and discrimination.

TD lifetime measurements, which are more intuitive, exploit the fact that the fluorescence emission is theoretically proportional to the number of molecules in the first excited state S_1, and as a result it decays exponentially [31]. In TD method, a short (relative to the fluorescence lifetime) light pulse (e.g., laser pulse) is used to repetitively activate the fluorescence molecule, and the emitted fluorescence is measured directly in time. To reconstruct the exponential decay, two most common techniques are used: time-gated single photon counting (TGSPC) [16, 17] and time-correlated single photon counting (TCSPC) [19, 20].

On the other hand, for FD FLIM, the intensity of the excitation light is continuously modulated at 10-200 MHz frequencies. A sinusoidally modulated light source is usually employed to excite the fluorescence molecule. Due to the (non-instant) fluorescence decay, the resulting sample emission is also sinusoidally modulated at the same frequency, but the phase has been shifted and the modulation amplitude is decreased.

Thus, the lifetime(s) is determined from the observed phase shift and the decrease in modulation amplitude.

2.3 Time-domain FLIM by TCSPC

Due to superior temporal resolution and recent developments in advance laser sources and CMOS cameras, TCSPC techniques [19] have become the gold standard within the FLIM community [7, 21, 22]. TCSPC can directly measure the actual decay curve with raw timing data. For a given number of photons detected from the sample, TCSPC delivers the better photon efficiency than other techniques. Moreover, TCSPC FLIM is widely compatible with the advanced microscopes, e.g., confocal and multiphoton laser scanning microscopes [7].

2.3.1 TCSPC

TCSPC can sufficiently record the photons and measure their time in the signal period, then build up a histogram based on the photon times [19]. Figure 2.3 illustrates the general principle of TCSPC. The sample is repetitively excited with a high frequency light source. During each signal period (measurement window), the first emitted photon (there are many signal periods without photons) is recorded. The arrival time of each detected photon is measured, and the photon count of the corresponding time bin is incremented by 1. By accumulating all the detected photons, the distribution of the detection times can be acquired. Since the probability density that a photon is emitted at time t is proportional to the exponential emission decay curve, the fluorescence decay histogram can be reconstructed based on the distribution curve (reconstructed fluorescence histogram). TCSPC can be very precise since it records photon signals with a high time resolution and a near-ideal efficiency [19].

Although the intrinsic fluorescence decay follows an exponential profile, as one can find from Figure 2.3 the reconstructed fluorescence histogram is not a simple exponential decay. In practice, the photon distribution is the convolution of the intrinsic fluorescence decay with the instrument response function (IRF) [19]. The IRF is defined as the pulse

signal the FLIM system records for an infinitely short fluorescence lifetime. More specifically, it is the convolution of the excitation pulse signal, the transit-time-spread of the detector, the pulse dispersion in the optical system, and the on-off transition time of the gating pulse [19].

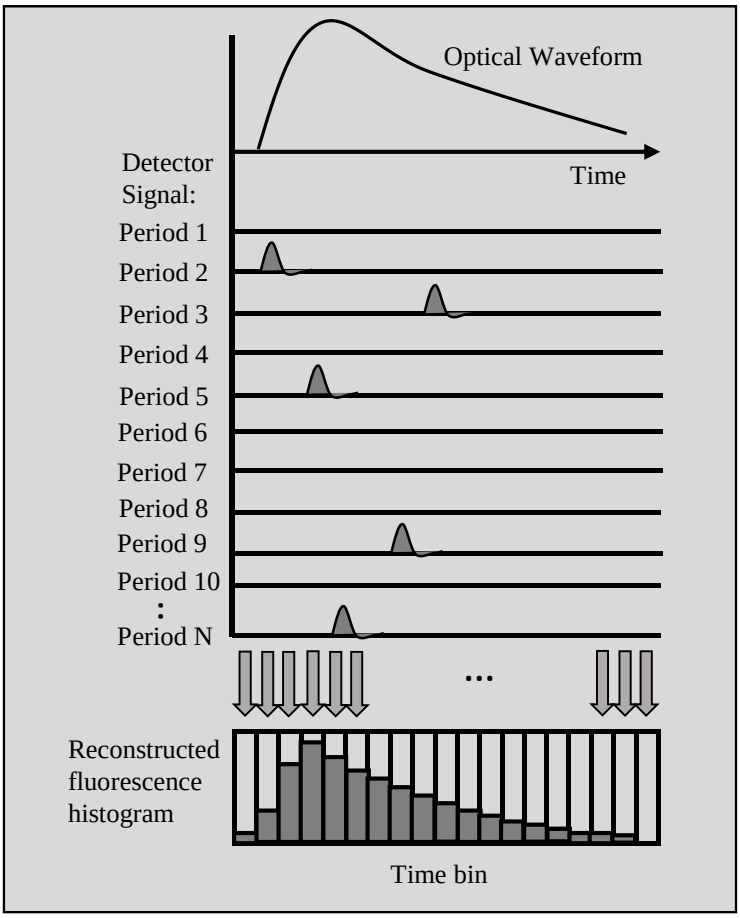

Figure 2.3 - General principle of TCSPC (Reproduced from [21]). The fluorescence sample is repetitively excited by a pulse light source. During each signal period, the first emitted photon is recorded. The arrival time of each detected photon is measured, and the photon count of the corresponding time bin is incremented by 1. By accumulating all the detected photons, the fluorescence decay histogram is reconstructed.

In current implementations of laser scanning microscopy, conventional TCSPC is fundamentally limited with respect to the photon counting rate. More specifically, conventional laser scanning TCSPC FLIM systems usually require several minutes to finish the data acquisition for one FLIM image [32]. In order to overcome this limitation,

multi-channel photomultiplier tube (PMT) based TCSPC instruments have been employed [33, 34]. Moreover, thanks to the dramatic development of CMOS technologies, time-to-digital convertor (TDC) arrays with integrated single-photon avalanche diodes (SPADs) have been developed for microscopy and spectroscopy techniques [23, 24]. A SPAD is a solid-state photodetector, where an avalanche current can be triggered by a photon when biased at above its breakdown voltage [35]. The latest multifocal multiphoton FLIM microscope, which parallelizes the excitation and detection process, has dramatically improved the acquisition rate of high resolution fluorescence lifetime imaging [22].

2.3.2 Instrument response function

In order to obtain the accurate fluorescence lifetimes, it is normally required to properly account for the instrument response function (IRF), which is the mapping between the incoming photon flux and the detected events.

For a FLIM system, each pixel contains a large number of time bins (channels) spread over the decay data. An appropriate model is needed to obtain fluorescence lifetimes from the decay data of each pixel. However, the time resolution of the FLIM measurement system is finite [36]. As a result, it is undeniable that the IRF affects the decay data. For most FLIM analysis procedures, the theoretical model function is first convoluted with the IRF, then compared with the decay data. During the fitting procedure, the parameters of the model are updated until they reach the best fit.

Unfortunately, it is very difficult to record an accurate IRF for a FLIM system. First, the dichroic beamsplitter of the microscope does not pass the excitation wavelength on to the detectors [36].

Moreover, the excitation beam path usually contains a laser blocking filter that cannot easily be removed. Additionally, a signal detected at the laser wavelength is usually contaminated by reflections and scattering in the optical system, and by fluorescence from the target.

The most vexing instrumental uncertainty in time-domain FLIM hence arises from the unknown IRF. The IRF corresponds mathematically to the shape of the emission that would be recorded by the detector if the fluorophore of interest had an infinitesimally short (δ-function) lifetime instead of its normal lifetime. The exact IRF profile therefore depends on the average shape and arrival time of the incident light pulses, any optical delays or aberrations, and the detector response (e.g., electronic timing jitter, wavelength-dependent broadening). The IRF could also be affected by contaminating fluorescent signals or other possible instrumental artefacts (addressed below). If the IRF could be robustly determined, straightforward deconvolution—or, more preferably, fitting based on iterative reconvolution [7, 37]—would reveal the true underlying fluorescence decay. Unfortunately, the exact shape of the IRF is difficult to reliably determine. As we show below, precise knowledge of the IRF is essential for accurate determination of lifetimes and population fractions.

2.3.3 Standard algorithms

To generate a lifetime image requires a FLIM analysis algorithm to process a large number of pixels, where each pixel contains a reconstructed fluorescence decay histogram generated by a TCSPC based data acquisition systems. Fast imaging systems, however, have significantly increased the data throughput and made the image analysis much more challenging than before.

FLIM algorithms can be classified into two categories: tail-fitting and IRF based fitting methods. As for the tail-fitting procedures, although traditional algorithms (such as maximum likelihood estimation (MLE) [37], least square method (LSM) [37, 38], and Global Analysis (GA) [39-41]) are able to deliver high performance results, they are generally slow. To tackle this problem, several fast non-iterative FLIM algorithms have been introduced, e.g., integral equation method (IEM) [42], center of mass method (CMM) [35, 43], phasor method (PM) [44], and bi-decay CMM (BCMM) [45].

On the other hand, for the IRF based methods, one of the most common techniques is the nonlinear least squares iterative reconvolution method (LSIR), which can be easily

adapted from the tail fitting least square method [46-49]. However, with traditional least square algorithms it is impossible to achieve real-time FLIM analysis due to heavily iterative computations. Several Laguerre expansion based deconvolution methods have been proposed [50, 51], and they are much faster than the reconvolution method. Also, robust Bayesian methods for FLIM analysis have been presented recently [52].

2.3.3.1 Tail-fitting methods

As shown in figure 2.4, for tail-fitting methods, the valid data starts from the peak of the measured histogram. That means the IRF is neglected, as in the model demonstrated by Leray et al [44]. Assume that the measured densities are $f\__S(t) = K \cdot \exp(-t/\tau)$ and $f\__B(t) = K \cdot [f_D \cdot exp(-t/\tau_F) + (1- f_D) \cdot exp(-t/\tau_D)]$ for single- and bi-exponential decays, respectively. τ, τ_F, τ_D are the lifetimes, K the pre-scalar, f_D the proportion of protein interaction. The assumptions allow a proper comparison between different algorithms.

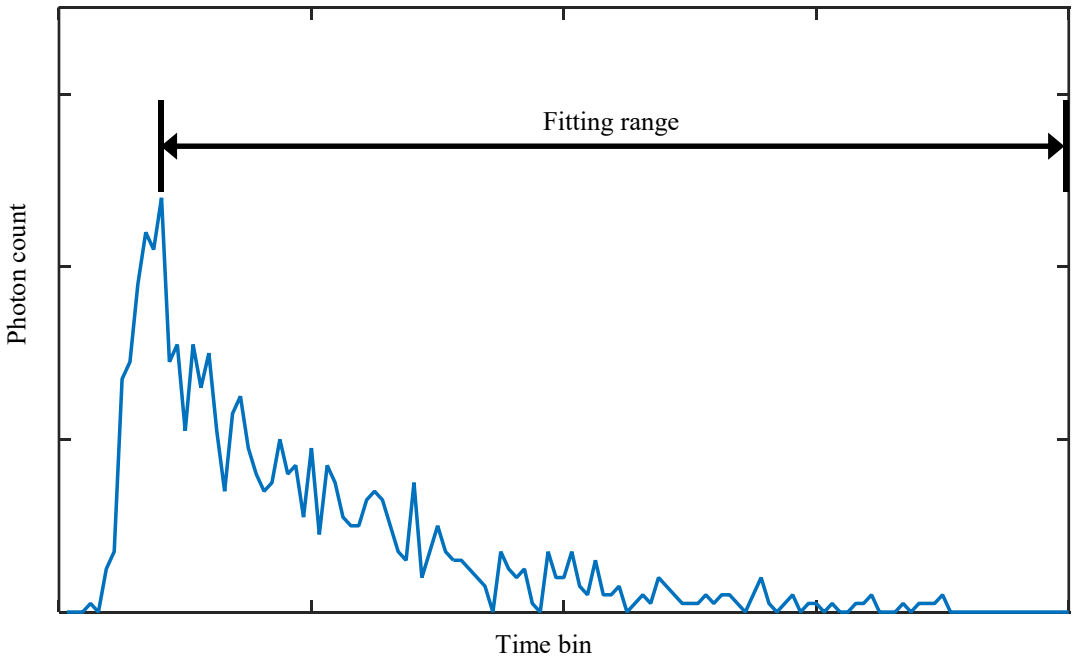

Figure 2.4 - The fitting range of tail-fitting methods, which starts from the peak of the histogram.

2.3.3.1.1 **IEM** [42]

Figure 2.5 shows a single-exponential fluorescence decay histogram $f(t)$ with M time bins.

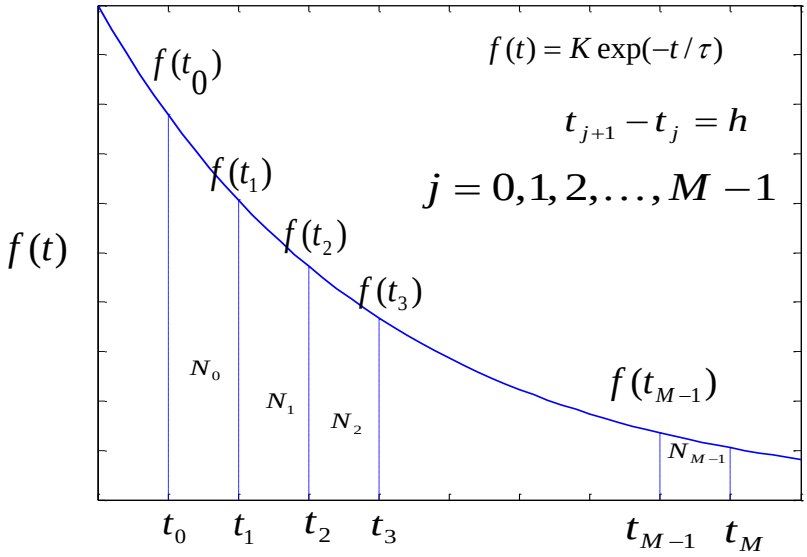

Figure 2.5 - A single-exponential decay and concept of IEM algorithm (Modified from [42]).

For a single-exponential decay, the lifetime constant is related to the area under the histogram function as

$$\tau(f_0 - f_{M-1}) = \int_{t_0}^{t_{M-1}} f(t)dt = \int_{t_0}^{t_{M-1}} K\exp(-t/\tau)dt \tag{2.5}$$

where $f_j = f(t_j)$. When Simpson's rule is applied to calculate the numerical integration, we can get

$$\begin{aligned} &\tau(f_0 - f_{M-1}) = \int_{t_0}^{t_{M-1}} f(t)dt \\ &\cong \frac{h}{3}(f_0 + 4f_1 + 2f_2 + \cdots + 4f_{M-2} + f_{M-1}) \end{aligned} \tag{2.6}$$

Then multiply equation 2.6 on both sides by the factor $(1\text{-}e^{-h/\tau})$ to get [42]

$$\begin{aligned}\tau_{IEM} &\cong \frac{\frac{h}{3}(f_0+4f_1+2f_2+\cdots+4f_{M-2}+f_{M-1})\cdot(1-e^{-h/\tau})}{(f_0-f_{M-1})\cdot(1-e^{-h/\tau})} \\ &\cong \frac{\frac{h}{3}(EN_0+4EN_1+2EN_2+\cdots+4EN_{M-2}+EN_{M-1})}{EN_0-EN_{M-1}} \\ &= \frac{h\sum_{j=0}^{M-1}(C_jEN_j)}{EN_0-EN_{M-1}} \\ &= \frac{h\sum_{j=0}^{M-1}(C_jN_j)}{N_0-N_{M-1}}\end{aligned} \tag{2.7}$$

where $EN_j=\int_{jh}^{(j+1)h} f(t)dt = f_j(1-e^{-h/\tau})$, $EN_j = N_j$ (the number of counts in the jth time bin), and $C_j=[1/3,4/3,2/3,...,4/3,1/3]$ is the coefficient of Simpson's rule (Romberg's integration coefficient $C_j=[1/2,1,1/2,...,1,1/2]$ can also be used).

2.3.3.1.2 **CMM** [29, 35]

For an object with a continuous distribution of mass density $f(r)$, the center of mass (CM) is described as [29, 35]

$$CM=\frac{\int rf(r)dV}{\int f(r)dV} \tag{2.8}$$

For a single-exponential function $f(t) = K\cdot\exp(-t/\tau)$ in the range $0\le t\le T$, where $T=(M-1)h$, we have

$$CM=\frac{\int_0^T tf(t)dt}{\int_0^T f(t)dt}=\tau-\frac{Te^{-T/\tau}}{1-e^{-T/\tau}} \tag{2.9}$$

When $T>>7\tau$, from equation 2.9 we can get

$$
\begin{aligned}
\tau_{CMM} &\cong \tau - \frac{Te^{-T/\tau}}{1-e^{-T/\tau}} = \frac{\int_0^T tf(t)dt}{\int_0^T f(t)dt} \\
&\cong \frac{\int_0^T \Delta t f(t)dt}{\int_0^T f(t)dt} = \frac{\int_{t_0}^{t_1}(t-t_0)f(t)dt + \cdots + \int_{t_{M-1}}^{tM}(t-t_0)f(t)dt}{\int_{t_0}^{t_M} f(t)dt} \\
&\cong \frac{\int_{t_0}^{t_1}(\frac{t_0+t_1}{2}-t_0)f(t)dt + \cdots + \int_{t_{M-1}}^{tM}(\frac{t_{M-1}+t_M}{2}-t_0)f(t)dt}{\int_{t_0}^{t_M} K\exp(-t/\tau)dt} \\
&= \frac{\sum_{j=0}^{M-1}\Delta t_j \int_{t_j}^{t_{j+1}} f(t)dt}{N_c} = \frac{\sum_{j=0}^{M-1}\Delta t_j N_j}{N_c} = \frac{\frac{N_c}{2}\sum_{j=0}^{M-1} jN_j}{N_c}h \\
&= (\frac{\sum_{j=0}^{M-1} jN_j}{N_c} + \frac{1}{2})h
\end{aligned}
\tag{2.10}
$$

where $\Delta t_j = t_j - t_0 + h/2 = (j + 1\text{-}1/2)h$ and N_c is the total photon count.

2.3.3.1.3 **PM** [44, 53]

For PM, assume that the measured density is $f(t) = K\cdot[f_D\cdot exp(-t/\tau_F) + (1-f_D)\cdot exp(-t/\tau_D)]$. An intensity histogram can be converted into $[u, v]$ coordinates, where u and v coordinates are the cosine and sine transforms of the density $f(t)$ respectively defined by

$$
u = \frac{\int_0^T f(t)\cos(\omega t)dt}{\int_0^T f(t)dt} \tag{2.11}
$$

$$
v = \frac{\int_0^T f(t)\sin(\omega t)dt}{\int_0^T f(t)dt} \tag{2.12}
$$

where ω is the laser repetition angular frequency. The fluorescence lifetime of the donor in presence of the acceptor τ_F and the fraction of interacting donor f_D can be analytically expressed as

$$\tau_F^P = \frac{1-u-v\tau_D\omega}{\omega(v-u\tau_D\omega)} \tag{2.13}$$

$$f_D^P = \frac{\tau_D(1+\tau_F^2\omega^2)(1-u-u\tau_D^2\omega^2)}{(\tau_F-\tau_D)(-1+u+u\tau_F^2\omega^2+\tau_F\tau_D\omega^2+u\tau_F^2\tau_D^2\omega^4)} \tag{2.14}$$

2.3.3.1.4 **BCMM** [45]

Assume that the fluorescence decay is $f(t) = K\cdot[f_D\cdot exp(-t/\tau_F) + (1- f_D)\cdot exp(-t/\tau_D)]$. Similar to PM, when τ_D is fixed, the lifetime of acceptor τ_F and the fraction of interacting donor f_D can be acquired by

$$\tau_F = \frac{\tau_D N - X}{\tau_D K - N} \tag{2.15}$$

$$f_D = \frac{\tau_D K - N}{K(\tau_D - \tau_F)} \tag{2.16}$$

where $N = \sum_{j=0}^{M-1}(C_j N_j), X = \sum_{j=0}^{M-1}(C_j t_j N_j), K = N_0 / h$, and $C_j, j=1, 2,...,M$, are coefficients of Simpson's or Romberg's integration rule.

Whereas, if τ_D is unknown, the lifetimes of donor τ_D, acceptor τ_F, and the fraction of interacting donor f_D can be given by

$$\tau_F = 0.5\cdot[G - \sqrt{G^2 - 4(N\cdot G - X)/K)}] \tag{2.17}$$

$$\tau_F = 0.5\cdot[G + \sqrt{G^2 - 4(N\cdot G - X)/K)}] \tag{2.18}$$

$$f_D = \frac{K\tau_D - N}{K(\tau_D - \tau_F)} \tag{2.19}$$

where $G = (K \cdot Y - N \cdot X)/(K \cdot X - N^2), Y = \sum_{j=0}^{M-1}(C_j \frac{t_j^2}{2} N_j)$, and N, X, K are the same as described earlier.

2.3.3.1.5 **LSM** [37, 38]

The nonlinear LSM consists in minimization of the cost function:

$$\begin{aligned} \chi^2 &= \sum_{j=0}^{M-1}(\frac{N_j - Y_j}{\sigma_j})^2 \\ &= \sum_{j=0}^{M-1}(\frac{N_j - Y_j}{\sqrt{N_j}})^2 \end{aligned} \tag{2.20}$$

where, $Y_j = \sum_{k=1}^{n}(A_k e^{-t_j/\tau_k})$, n is the number of lifetime components (e.g., for single- and bi-exponential decay, n is 1 and 2 respectively), and variance σ_j^2 is equal to N_j as photon counts follow Poisson distributions.

The minimization process is an iterative procedure, and some optimization algorithms such as the Levenberg–Marquardt algorithm [54] can be used to find the results of unknown lifetime parameters.

2.3.3.1.6 GA [39-41]

Previously described FLIM algorithms treat each pixel (histogram) independently, while GA assumes that there are fixed lifetimes with the same segments (a group of pixels) across the image. This approach employs a similar fitting approach as LSM, but has the cost function:

$$\chi^2 = \sum_{i=1}^{N_{GA}} \sum_{j=0}^{M-1} \left(\frac{N_{i,j} - Y_j}{\sigma_{i,j}}\right)^2 = \sum_{i=1}^{N_{GA}} \sum_{j=0}^{M-1} \left(\frac{N_{i,j} - Y_j}{\sqrt{N_{i,j}}}\right)^2 \tag{2.21}$$

where, N_{GA} is the number of pixels in a segment.

GA is capable of exploiting the spatial variation of contributions across the segments, and giving stronger constraints on the lifetime estimates which allows GA to produce more accurate lifetime results than LSM. However, GA is computationally heavier than LSM.

2.3.3.2 IRF based methods

On the other hand, for IRF based methods, the whole measured histogram is used for lifetime calculation as shown in figure 2.6. As we have learnt the measured fluorescence decay histogram $y(t)$ is equivalent to the sample's intrinsic fluorescence decay convoluted with the excitation profile (i.e., IRF) $E(t)$ [5]:

$$y(t) = E(t) \otimes I(t) = \int_0^t E(t - z) \cdot I(z) dz \tag{2.22}$$

where $I(t)$ is the intrinsic fluorescence density function (theoretical exponential decay function).

For a TCSPC based FLIM system, the decay signal Eq. (2.23) can be re-written as a discretized model:

$$y(k) = \sum_{i=0}^{k} E(k-i) \cdot I(i) + \varepsilon(k), \; k = 0,1,2,\ldots, M-1 \tag{2.23}$$

where $\varepsilon(k)$ is the Poisson noise introduced during the measurement, and M is the number of time bins.

For IRF based FLIM analysis, we assume the decay is bi-exponential:

$$I(k) = Kf_D \exp[-kh/\tau_F] + K(1-f_D)\exp[-kh/\tau_D] \quad (2.24)$$

where K is pre-scalar, f_D is the proportion, h is the bin width, and τ_F, τ_D are the lifetimes.

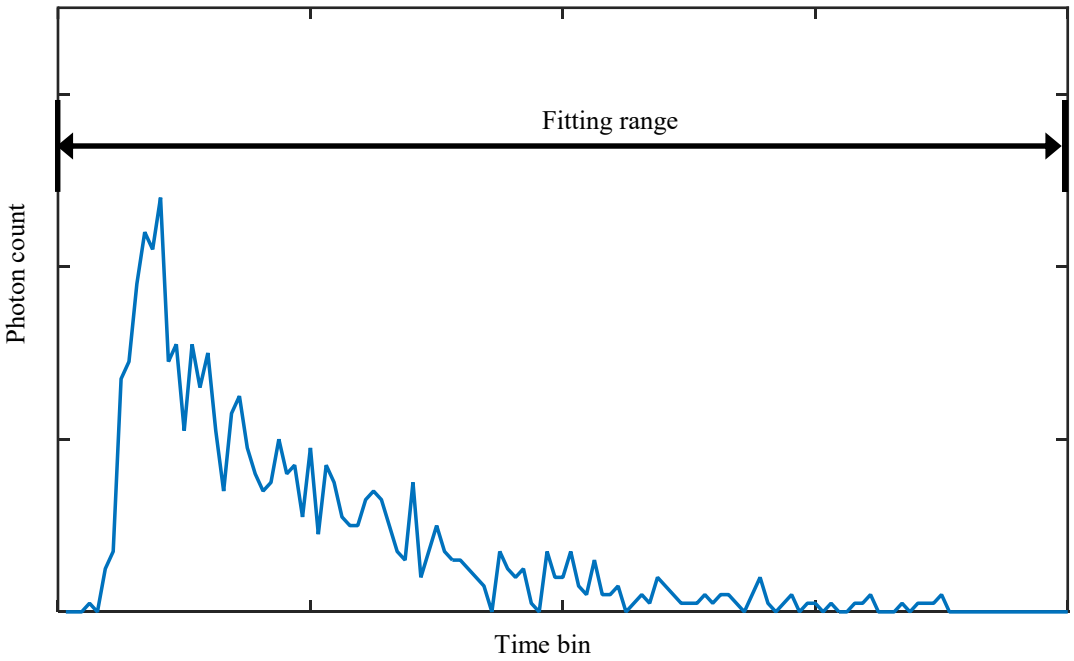

Figure 2.6 - The fitting range of IRF based methods.

2.3.3.2.1 **LSIR** [47-49]

Although LSIR is a slow analysis method, it is one of the most reliable and commonly used methods [47-49]. Theoretical exponential decay can be obtained by applying the trial parameter set α = [K, f_D, τ_F, τ_D] to E.q. (2.25), and the calculated decay can be constructed by convoluting the IRF with the exponential decay. The IRF can be obtained either from synthesized or experimental data, and it remains unchanged during the lifetime analysis. The optimization method tries to minimize the cost function by updating the parameters α. The lifetime results can be acquired from α, when the optimization process converges. The cost function, which defines the error between the measured data and the calculated decay, is given by

$$\chi^2 = \sum_{j=0}^{M-1} \frac{[N_j - y(t_j)]^2}{\sigma_j^2} = \sum_{j=0}^{M-1} \frac{[N_j - y(t_j)]^2}{N_j} \tag{2.25}$$

where variance σ_j^2 is equal to N_j, as photon counts follow a Poisson distribution.

2.3.3.2.2 **LSD-LE** [51]

Traditional reconvolution algorithms make it impossible to achieve real-time FLIM analysis due to heavily iterative computations. Several Laguerre expansion based deconvolution methods have been proposed [50, 51], and they are much faster than LSIR. For example, an improved computationally effective Laguerre expansion based least squares deconvolution method (LSD-LE) [51] has been proposed recently. In this method, after acquiring the IRF, theoretical exponential decay is retrieved by using a Laguerre expansion based deconvolution method. Then lifetime parameters are calculated from the obtained exponential decay by LSM.

The discrete intrinsic fluorescence density function $I(k)$ can be expanded by an ordered set of discrete-time Laguerre basis functions (LBFs) [55]:

$$\hat{I}(k) = \sum_{l=1}^{L} c_l b_l(k;\beta), \quad k = 0,1,2,\ldots,M-1 \tag{2.26}$$

where c_l is the lth expansion coefficient, and $b_l(k;\beta)$ is determined by the Laguerre dimension L and the scale β.

By substituting Eq. (2.26) into (2.23), we can get:

$$y(k) = \sum_{l=1}^{L} c_l v_l(k) + \varepsilon(k), \quad v_l(k) = \sum_{i=1}^{k} E(k-i) b_l(k;\beta) \tag{2.27}$$

The goal of deconvolution is to estimate c_l. Here we rewrite Eq. (2.27) as

$$\boldsymbol{y} = \boldsymbol{V}\boldsymbol{c} + \boldsymbol{\varepsilon} \tag{2.28}$$

Liu et al. proposed a constrained LSD-LE called CLSD-LE [56], by giving

$$\hat{\boldsymbol{c}} = (\boldsymbol{V}^{\boldsymbol{T}}\boldsymbol{V})^{-1}(\boldsymbol{V}^{\boldsymbol{T}}\boldsymbol{y} - \boldsymbol{D}^{\boldsymbol{T}}\hat{\lambda}) \tag{2.29}$$

where $\hat{\lambda}$ is the solution to non-negative least square problem

$$\underset{\lambda \in \boldsymbol{R}^{N-3}}{\text{minimize}} \left\| (\boldsymbol{V}^{\boldsymbol{T}}\boldsymbol{V})^{-1}(\boldsymbol{V}^{\boldsymbol{T}}\boldsymbol{y} - \boldsymbol{D}^{\boldsymbol{T}}\lambda) \right\|^2, \text{ subject to } \lambda \geq 0 \tag{2.30}$$

where $\mathbf{D}=\mathbf{D}^{(3)}\mathbf{B}$ is the third-order forward finite difference matrix for the LBFs $\mathbf{B}=[b_l|l=1,2,3,\ldots,L]$. Then the recovered intrinsic decay can be acquired form Eq. (2.26) as

$$\hat{I}(k) = \sum_{l=1}^{L} \hat{c}_l b_l(k;\beta) \tag{2.31}$$

After applying Eq. (2.31) to Eq. (2.24), we can get

$$I(k) = Kf_D \exp[-kh/\tau_F] + K(1-f_D)\exp[-kh/\tau_D] = \sum_{l=1}^{L} \hat{c}_l b_l(k;\beta) \tag{2.32}$$

Then unknown parameters K, f_D, τ_F, τ_D can be obtained by using LSM.

2.4 FLIM applications

FLIM is a key fluorescence microscopy technique to deliver information about the environment. Due to the fact that FLIM is independent of the local fluorophore concentration and excitation intensity, it is capable to observe photophysical events that are difficult or impossible to detect with fluorescence intensity imaging [57]. There are a variety of technical solutions of lifetime imaging in microscopy. One prominent FLIM application is to study protein interactions and conformational changes by combining

with FRET, and FLIM is also used to image pH, O_2, Ca^{2+}, ion concentrations, viscosity, temperature and refractive index, all at the cellular level, as well as cell and tissue auto-fluorescence.

FLIM technologies have been widely combined with Förster resonance energy transfer (FRET), and FLIM-FRET technologies are complementary methodologies that can be applied to advanced quantitative analyses [58]. FRET is a well-established technique to study molecular interactions with high spatial and temporal specificity. FLIM-FRET has been used for brain tumor image-guided surgery [12], studies of the causes for Alzheimer's disease [59]), cancer diagnosis [11], cancer therapies [6, 7], drug developments, study of drug efficacy [13], and clearance in tumors [14].

FLIM has also been used for mapping the intracellular pH (pHi). pHi is one of the most prominent features in living organisms, which affects many physiological processes (e.g., ion transport, cellular metabolism and cell cycle control) [60]. Dramatic improvement of FLIM has allowed it to quickly and accurately generate pHi based images, and FLIM-pHi has become a powerful tool for biochemistry and physiology.

For a large number of fluorophores, oxygen is an efficient fluorescence quencher, especially those of a longer fluorescence lifetime [61]. It has been observed that oxygen can cause strong quenching for the phosphorescence of organic complexes of ruthenium, europium, platinum and palladium. Unlike intensity based fluorescence imaging of oxygen in cells, lifetime measurements would not require a calibration of the intensity of the probe unquenched by oxygen [57].

In living cells and organisms ions play a significant role. For cell biologists and physiologists, it is very important to map and measure ion concentrations and to dynamically observe changes and fluctuations [57]. Also, ions, especially Ca2+ and Cl−, can cause quenching of the fluorescence or phosphorescence, both of which are important to the function of the neuronal system [61].

Viscosity, which affects the mobility of the cell membrane and thus the active transportation of bioactive molecules through the membranes, plays paramount roles in

biological systems (e.g., phospholipid bilayer systems) [62]. Viscosity measurement by FLIM has been demonstrated by Kuimova et al. [63] and Levitt et al. [64].

Another application of FLIM is to map the temperature in living cells in combination with special temperature-sensitive polymers [57]. Intracellular temperature can fundamentally regulate cellular functions, and also influences biochemical reactions inside a cell [13].

2.5 Summary

FLIM produces spatially resolved images of lifetime of the excited state of fluorophores, providing a deeper insight into the environment and the structure of the fluorophore. It has been recognized as a powerful imaging tool in biological sciences. Due to the dramatic development of high-speed data acquisition techniques of FLIM, measurements of fluorescence lifetime in live cells have become feasible. These properties have encouraged more and more applications of FLIM technologies, among which one prominent application is the identification of FRET to study protein interactions. However, for current FLIM systems, high-speed or real-time FLIM applications (the data throughput of the latest 2-D SPAD arrays can easily exceed tens of gigabit/s [65]) is still staggering, and lifetime analysis has become a bottleneck.

Chapter 3 Graphics processing units and FLIM analysis

3.1 Overview

Traditional FLIM systems, which use the CPU as the data processor, are relatively slow. Moreover, for FLIM analysis, each pixel is independent and lifetime calculation is based on the data of a single histogram. In order to realize high-speed or real-time FLIM analysis, it is necessary to introduce parallel computing technologies to FLIM application. Among parallel processors, graphics processing units (GPUs) are very well suited for FLIM analysis. GPUs have highly parallel architecture (contain thousands of parallel cores), high floating-point performance and memory bandwidth, and are more energy efficient than CPUs. This chapter begins with the brief introduction of parallel computing, along with the description of GPU computing. Then, an overview of the Compute Unified Device Architecture (CUDA) and the benefits of using GPU for FLIM analysis are described. Moreover, the GPU implementations of standard algorithms will be demonstrated, as well as some suggestions about how to optimize GPU programming. Finally, results are presented regarding the performance of GPU accelerated FLIM analysis with standard algorithms.

3.2 Parallel Computing

In terms of processing speed, it is now clear that silicon based processor chips are reaching their physical limits, fundamentally the speed of light and quantum effects[66]. An effective solution to overcome this grand challenge is to connect multiple processors working in coordination with each other. Parallel computing is a computer science

discipline, and it deals with the system architecture and software issues related to the concurrent execution of applications. During the past several decades, it has been an area of active research interest and application. It is not only the focus of high performance computing, but also is now emerging as the prevalent computing paradigm.

3.2.1 Brief history of Parallel Computing

The interest in parallel computing began in the late 1950's, with advancements surfacing in the form of supercomputers throughout the 60's and 70's. These supercomputers contained multiprocessors, working side-by-side on shared data. Starting in the mid 1980's, massively parallel processors (MPPs) gradually came to dominate the top end of computing, especially with the ASCI Red supercomputer computer in 1997 breaking the barrier of one trillion floating point operations per second.

On the other hand, from the late 80's, clusters came to compete and eventually displace MPPs for many applications. A cluster is built from large numbers of off-the-shelf computers connected by an off-the-shelf network. Nowadays, clusters have become the workhorse of scientific computing, and are the dominant architecture in data centers.

In the current information age, parallel computing is becoming mainstream based on multi-core processors. Overall processing performance of desktop and laptop systems have been increased by adding additional CPU cores. This is because increasing performance through parallel processing can be far more energy-efficient than increasing microprocessor clock frequencies. Also, higher clock frequencies have heat dissipation issues. Parallel processing is especially essential for a world that is increasingly mobile and energy conscious.

According to the level at which the hardware supports parallelism, parallel computers can be classified into 4 categories: multi-core computing, symmetric multiprocessing, distributed computing, and specialized parallel computers.

3.3 Graphics processing units

As one of the specialized parallel devices, the GPU has been widely used in embedded systems, mobile phones, personal computers, workstations, and game consoles. Although GPUs were originally designed for handling computation only for computer graphics, they have been used to perform computation in applications traditionally handled by the CPU. General-purpose computing on GPUs only became practical and popular after about 2001.

3.3.1 GPUs as programmable parallel processors

In over a decade, GPUs have rapidly evolved from devices that simply implemented the traditional fixed-function 3D graphics pipeline towards flexible, programmable, general-purpose computational engines.

It was only after circa. 2001 that general-purpose GPUs became practical and popular, with the advent of both programmable vertex shaders and floating point support on graphics processors. In particular, GPUs are able to act with native speed and support on problems involving matrices and/or vectors. In 2001, the scientific computing community started to test the new hardware with matrix multiplication routines.

Since 2003, the semiconductor industry has settled on two main trajectories for designing microprocessor: multicore trajectory and many-core trajectory [67]. The multicore trajectory, which began with two-core processors, focuses on maintaining the execution speed of sequential programs while moving into multiple cores. On the other hand, the many-core trajectory seeks to explore more on the execution throughput of parallel applications, using a large number of far smaller cores. As a many-core processor, GPUs have led the race of floating-point performance and memory bandwidth, as illustrated in figure 3.1 and figure 3.2 [68]. During the past decade, the performance of GPUs has been improved dramatically, significantly higher than CPUs performance. As of 2016, many-core GPUs are able to achieve 10-fold higher peak single-precision floating-point calculation throughput than multicore CPUs. The reason behind the discrepancy in

floating-point capability between the CPU and the GPU lies in the differences in the fundamental design philosophies. More specifically, the GPU is specialized for compute-intensive and highly parallel computation, therefore more transistors are devoted to data

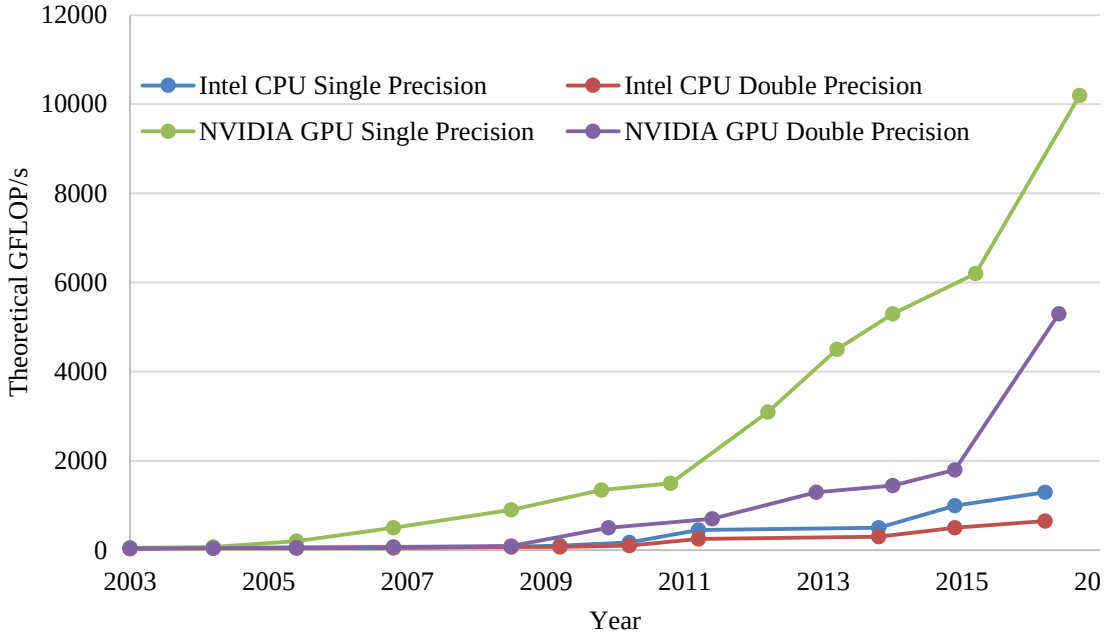

Figure 3.2 - Evolution of CPU and GPU floating-point performance (Reproduced from [68]). During the past decade, the performance of GPUs has been improved dramatically, which is significantly higher than CPUs performance.

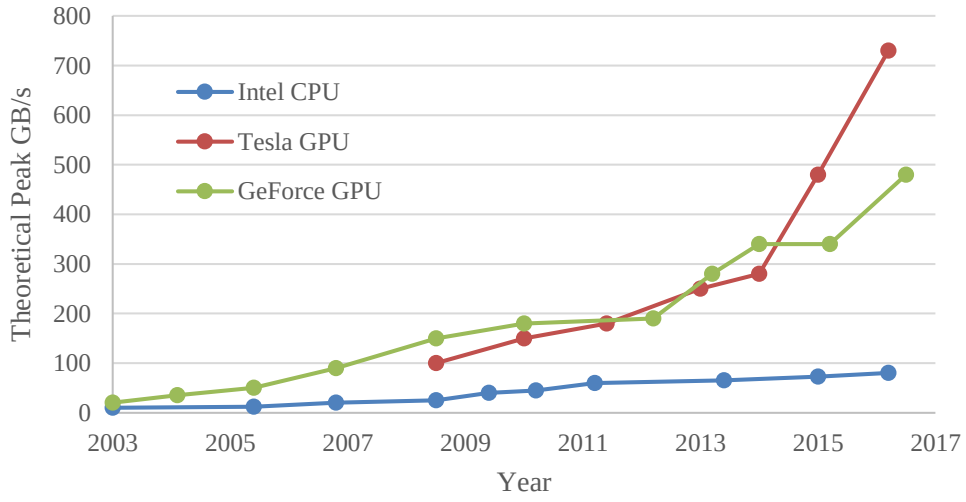

Figure 3.1 - Evolution of CPU and GPU memory bandwidth (Reproduced from [68]). Currently, the memory bandwidth of GPUs is more than 7-fold higher than CPUs'.

processing rather than data caching and flow control, as illustrated in figure 3.3.

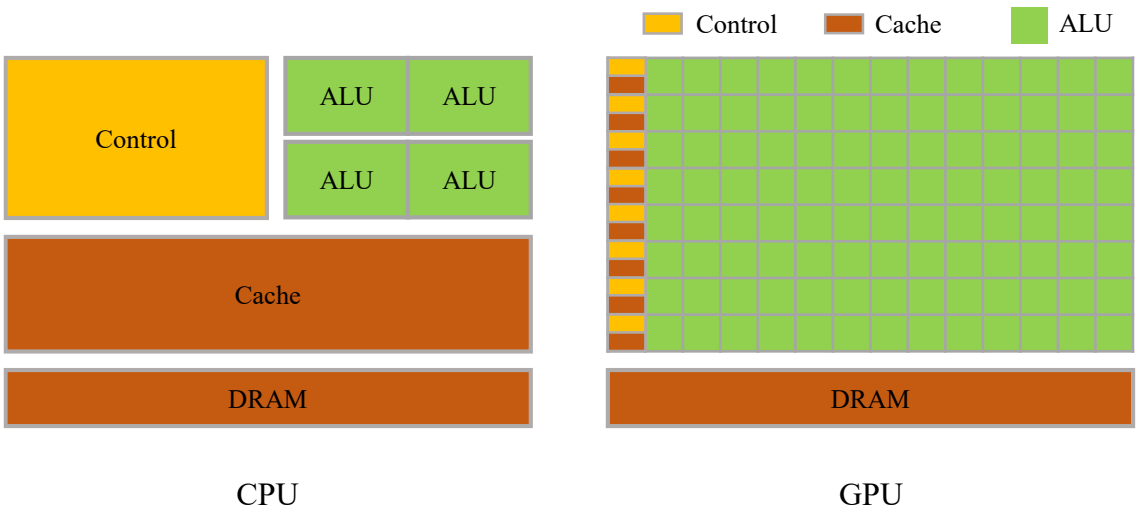

Figure 3.3 - Design philosophies of CPUs and GPUs (Reproduced from [68]). Compared with CPU, more transistors are devoted to data processing rather than data caching and flow control for GPU.

However, it should be clear that GPUs are designed as numeric computing engines, so they do not have as good performance in some tasks for which CPUs are designed. As a result, most applications will use both CPUs and GPUs, where the sequential parts will be executed on the CPU and GPUs will be dedicated to numerically intensive parts. More specifically, the processing flow of GPU applications contains GPU data flow, GPU program execution flow, and CPU copies of results from GPU memory [69]. As demonstrated in figure 3.4, input data is usually copied from CPU memory to GPU memory through the PCIe bus and it is always done by the CPU. The CPU is also responsible for launching a GPU kernel, then the GPU caches data on chip for performance and executes the program, as shown in figure 3.5. Then Giga Tread^{TM}, which is the warp (threads in groups of 32 parallel threads called warps) scheduler, distributes warps to its execution units [70, 71]. Figure 3.6 illustrates that after the GPU finishes computation, the results will be copied back to the CPU memory, and this is also usually done by the CPU.

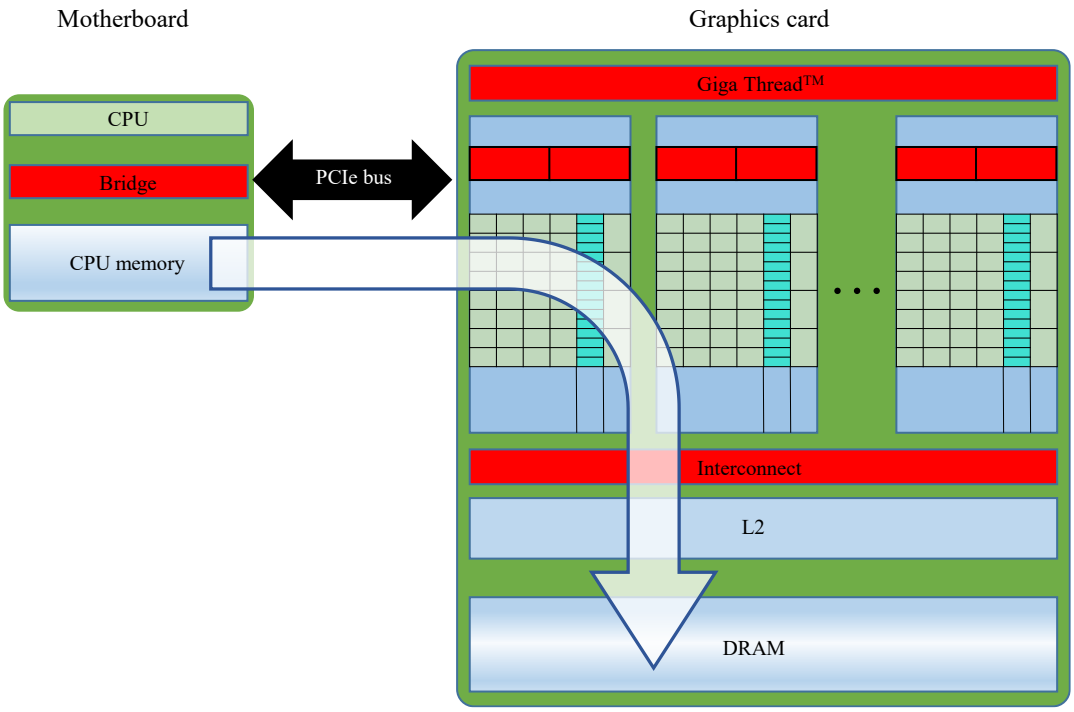

Figure 3.4 - Diagram of GPU data flow (Reproduced from [69]). Input data is copied from CPU memory to GPU memory through PCIe bus and it is always done by the CPU.

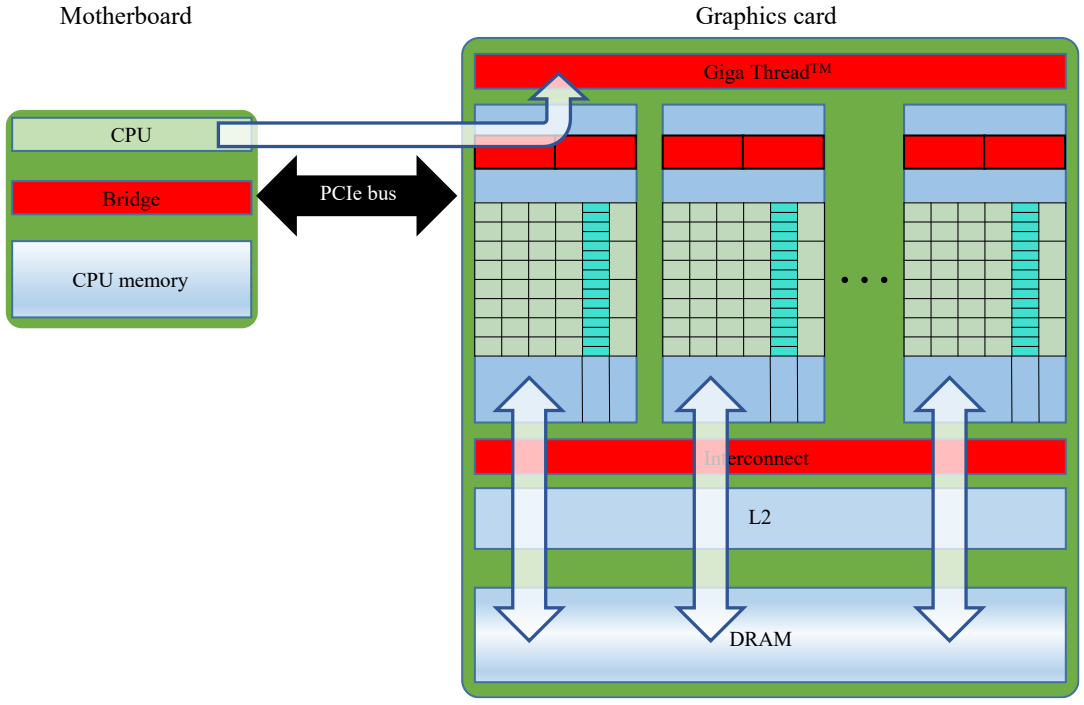

Figure 3.5 - Diagram of GPU program execution flow (Reproduced from [69]). CPU is responsible for launching a GPU kernel, then the GPU caches data on chip for performance and executes the program.

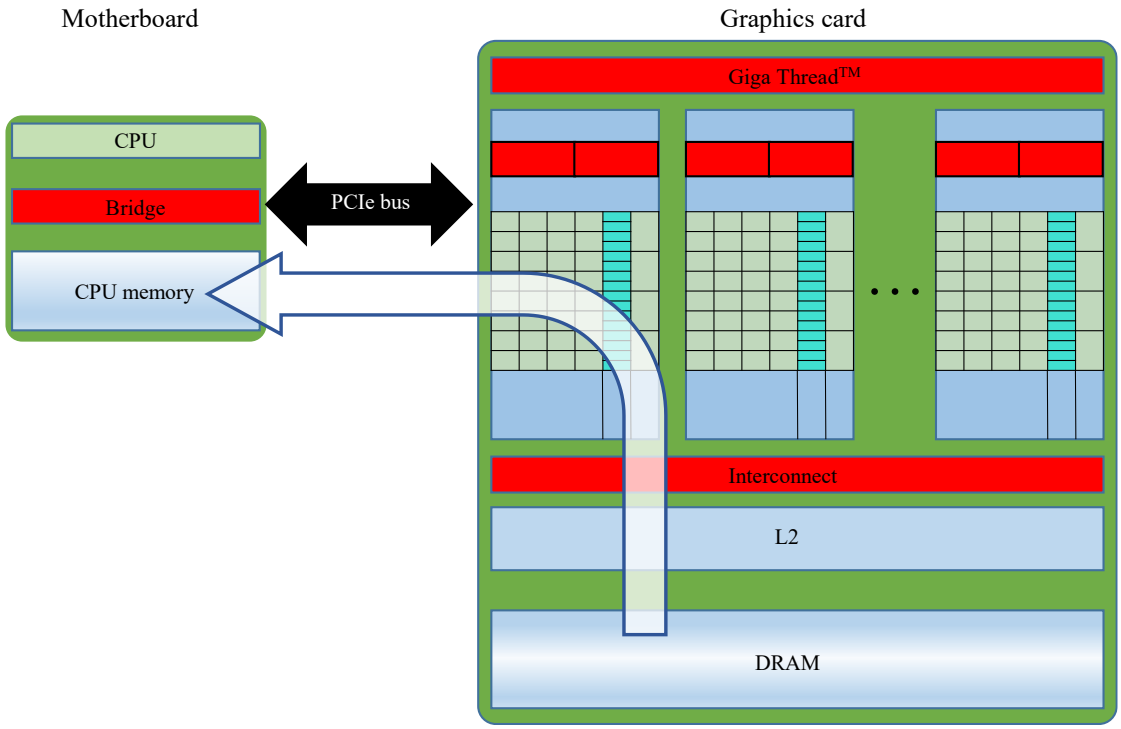

Figure 3.6 - CPU copies results from GPU memory (Reproduced from [69]). The results of GPU computation are sent back to the CPU memory, and it is also usually done by the CPU.

3.3.2 GPU architecture

The general NVIDIA GPU architecture (since GPU was first introduced by NVIDIA corporation [72] and it is still the major manufacture of GPUs, we use GPU standards provided by NVIDIA in our research) is built around a scalable array of multithreaded Streaming Multiprocessors (SMs). Each SM contains several Streaming Processors (SPs), and is designed to execute hundreds of threads concurrently. A unique architecture called single-instruction, multiple-thread (SIMT) is employed to manage such a large amount of threads. The SM SIMT unit creates, manages, schedules, and executes warps. Instructions within a single thread are pipelined to leverage instruction-level parallelism, as well as thread-level parallelism extensively through simultaneous hardware multithreading. In contrast with multi-core CPUs, a GPU consists of a high number of fragment processors with high memory bandwidth, and achieves higher parallel code performance.

Compared with CPUs, GPUs have more transistors that are devoted to data processing rather than data caching and flow control, so they are more suitable for compute-intensive, highly parallel computation. To be more specific, GPUs are especially well-suited to address problems for which the same instruction is executed on many data elements in parallel and have high arithmetic intensity. The requirement of sophisticated flow control is lower for GPUs, and the memory access latency can be hidden with calculations instead of big data caches.

Since the first GPU (NVIDIA GeForce 256) was introduced in 1999, NVIDIA has successfully proposed five GPU microarchitectures: 1) Tesla architecture, which is NVIDIA's first microarchitecture implementing the unified shader model, was introduced in 2006, and was used in the GeForce 8800 GTX (G80), 2) Fermi architecture was launched in 2010, and was the primary microarchitecture used in the GeForce 400 series and GeForce 500 series, 3) Kepler architecture, debuted in 2012, is NVIDIA's first microarchitecture to focus on energy efficiency, and most GeForce 600 series, most GeForce 700 series, and some GeForce 800M series GPUs were based on Kepler, 4) Maxwell architecture was introduced in 2014, in the GeForce GTX 750, 5) Pascal architecture, debuted in April 2016, in the GPU100 chip, 6) Volta architecture has been announced for Tesla V100 GPU in May 2017.

Although every new architecture introduces new features and focuses on different applications, they all share common architectural characteristics. Figure 3.7 demonstrates the architecture of a typical GPU. The GPU consists of an array of highly threaded SMs. As one can find from figure 3.7 (this figure was reproduced from [68]), each building block contains two SMs (the number of SMs in a building block varies from one generation of GPUs to another). Also, each GPU has its own graphics double data rate (GDDR) DRAM, which is known as global memory. Different from system DRAMs on the CPU motherboard, GPU DRAMs are essentially the frame buffer memory, and they function as very-high-bandwidth, off-chip memory. The SMs connect with the DRAMs via an interconnect network. Within a SM, there are a number of SPs that share control logic and instruction cache. Each SM also contains special function units (SFUs), instruction and constant caches, a multithreaded instruction unit, and a shared memory.

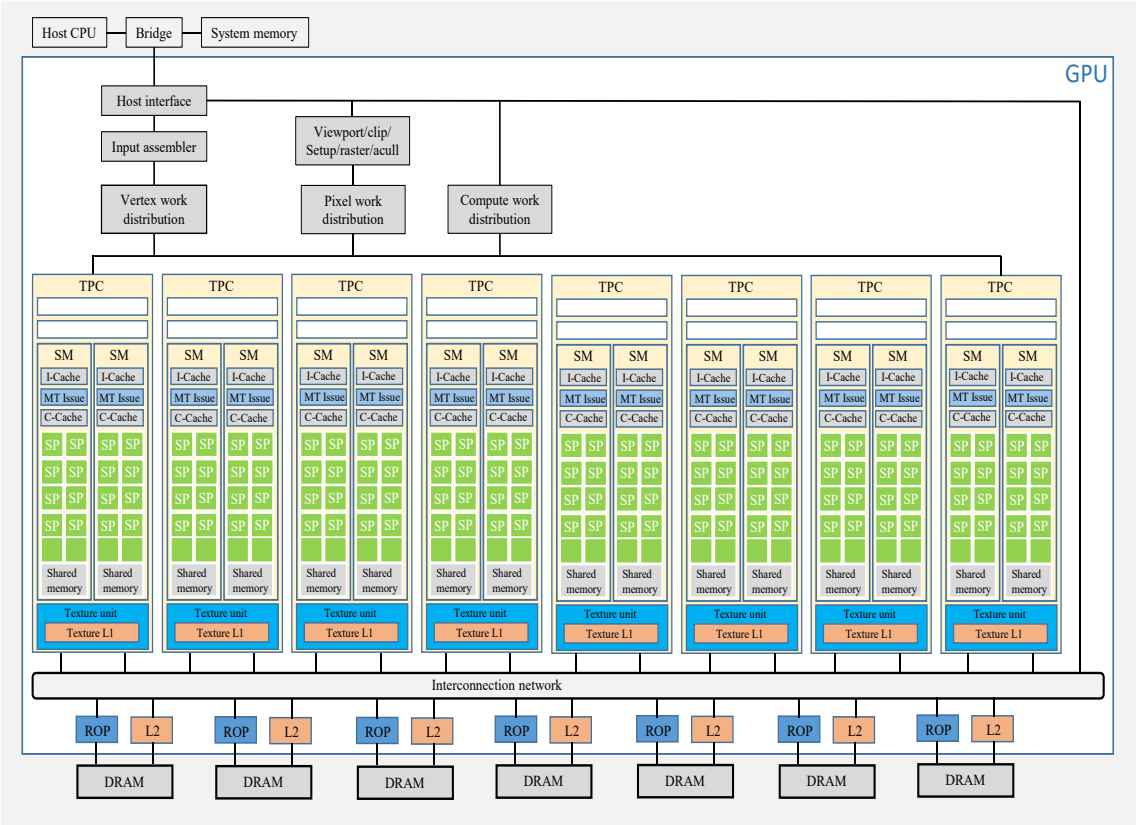

Figure 3.7 - A typical architecture of GPU (Reproduced from [68]). TPC: texture/processor cluster; SM: streaming multiprocessor; SP: streaming processor; ROP: raster operation processor. Each building block contains two SMs (the number of SMs in a building block varies from one generation of GPUs to another). Also, each GPU has its own graphics double data rate (GDDR) DRAM, which is known as global memory. The SMs connect with the DRAMs via an interconnect network. Within a SM, there are a number of SPs that share control logic and instruction cache. Each SM also contains special function units (SFUs), instruction and constant caches, a multithreaded instruction unit, and a shared memory.

3.4 CUDA overview

In 2006, NVIDIA introduced the compute unified device architecture (CUDA), a general purpose parallel computing platform and programming model that leverages the massively parallel processing power of NVIDIA GPUs to solve many complex computational problems in a more efficient way than on a CPU [68]. The CUDA architecture is a parallel computing architecture that enables the performance of NVIDIA's world-renowned GPU technology for general purpose computing.

With the CUDA architecture and software environment, a variety of industries and researches have enjoyed a great deal of success, in areas such as medical imaging and environmental science, creating breakthrough applications in areas such as image recognition and real-time HD video playback and encoding, and deep learning applications. CUDA comes with a software environment, where developers can choose to express the parallelism in high-level languages such as C, C++, Fortran, and OpenACC.

3.4.1 CUDA architecture

As shown in figure 3.8, the general CUDA architecture consists of the following basic ingredients [73, 74]:

1. CUDA parallel compute engines inside NVIDIA GPUs.
2. Operating system kernel-level support for hardware initialization, configuration, etc.
3. User-mode driver, which provides a device-level API (application programming interface) for developers.
4. Instruction Set Architecture named PTX (Parallel Thread Execution) for parallel computing kernels and functions.

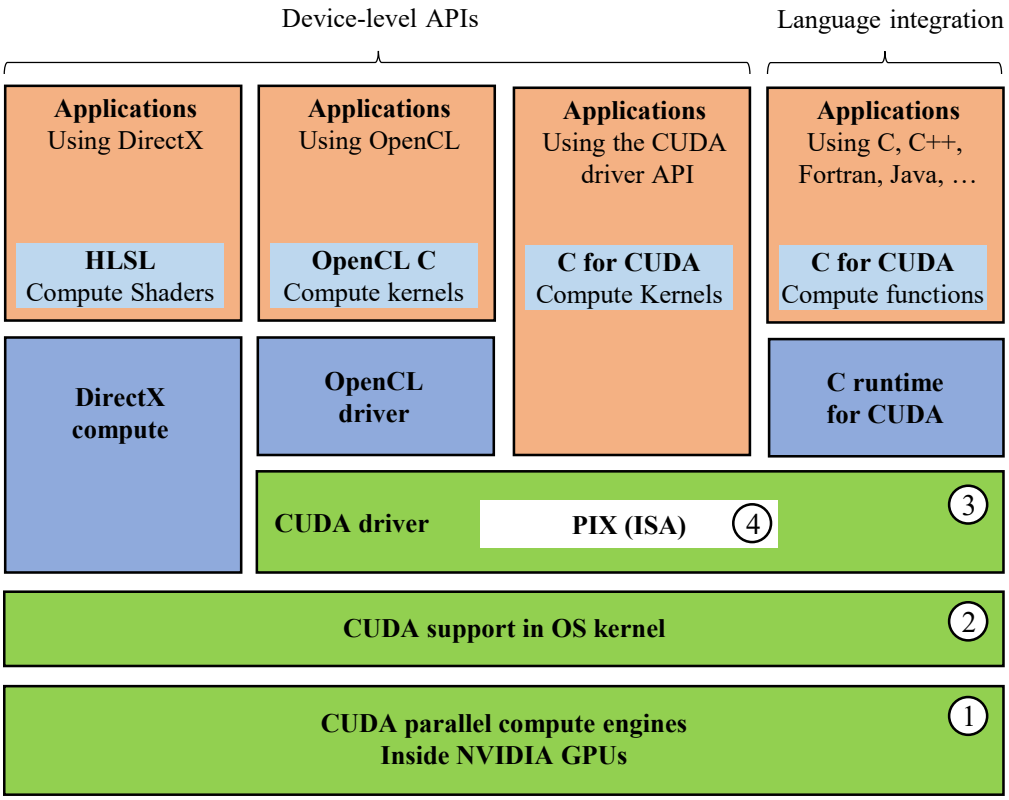

Figure 3.8 - CUDA architecture (Reproduced from [73]). The general CUDA architecture consists of the following basic ingredients: 1. CUDA parallel compute engines; 2. Operating system kernel-level support; 3. User-mode driver; 4. Instruction Set Architecture named PTX (Parallel Thread Execution)

3.4.2 Programming model

For a CUDA application, the computing system consists of a host that is a CPU, such as an Intel architecture microprocessor, and one or more GPUs with massively parallel processors. In modern computational intensive applications, program sections often exhibit a rich amount of data parallelism. The CUDA devices accelerate the execution of this portion of data parallelism. CUDA C is essentially C/C++ with a few extensions, and one can implement a parallel algorithms with it as easily as writing a C program.

3.4.2.1 Data parallelism

Data parallelism is a form of parallelization across multiple processors in parallel computing environments. It is a program property where many arithmetic operations can

be safely performed on the data structures concurrently [67]. In data parallel operations, multiple threads can operate on different segments concurrently, because the source is partitioned. For example, in matrix multiplication (A×B=C) each element of the product matrix is generated by performing a dot product between a row of input matrix A and a column of input matrix B. These dot products (for computing different C elements) can be simultaneously performed.

3.4.2.2 Program structure

A CUDA program, which supplies a single source code encompassing both host and device code, is executed on both the CPU and a GPU. It contains one or more phases. The phases that exhibit little or no data parallelism are usually executed on the CPU, and the phases that exhibit rich amount of data parallel computations are executed on the GPU. For a CUDA program, a special compiler called NVIDIA C Compiler (NVCC) is needed, which separates the host code and device code. To be more specific, the host code is compiled with the host's standard C compilers, whereas the device code is compiled by NVCC.

3.4.2.3 Kernels

Kernels are C functions that are compiled by NVCC and executed on the GPU device. Unlike regular C functions which are executed only once, kernels are executed N times in parallel by N different CUDA threads.

A kernel is highlighted by the __global__ declaration specifier, and the kernel's execution configuration (i.e., the number of CUDA threads of a given kernel call) is specified inside triple-angle-brackets:

```
kernel_name <<<grid, block>>> (argument list);
```

The execution configuration defines how the threads will be scheduled to run on the GPU. Figure 3.9 illustrates the execution of a typical CUDA program. First a CUDA program is executed in a CPU. Then a kernel function is launched, and the execution is moved to

a GPU by generating a large number of threads. A kernel is executed in a grid, which contains blocks, these blocks again contain threads, and the number of blocks and threads are defined by kernel arguments.

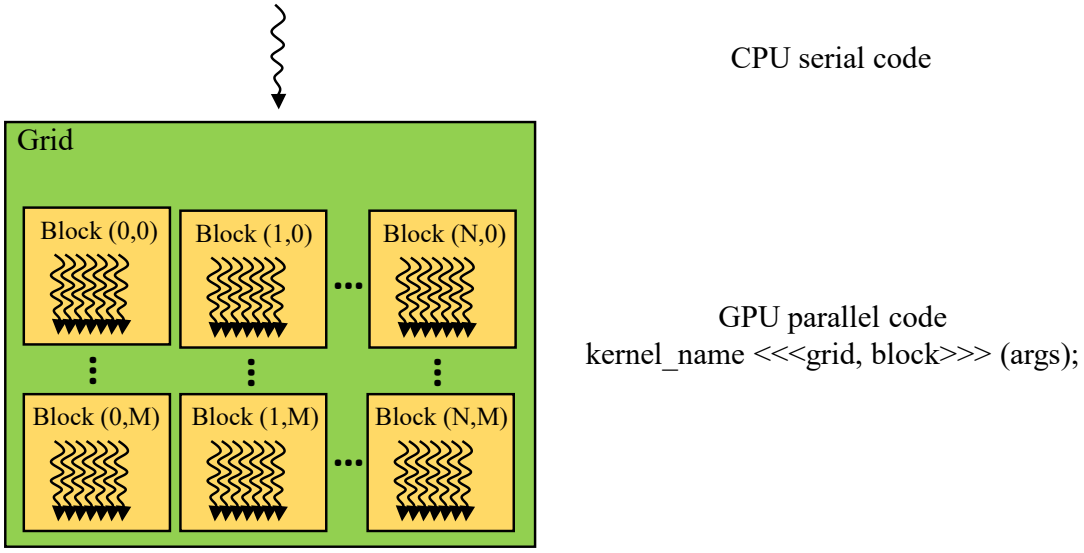

Figure 3.9 - Execution of a CUDA program. First a CUDA program is executed in a CPU. Then a kernel function is launched, and the execution is moved to a GPU by generating a large number of threads.

3.4.2.4 Thread Hierarchy

A CUDA thread is the basic execution unit in CUDA programming. Threads can be identified using a one to three dimensional index with a 3-component vector threadIdx. A thread block contains a one, two, or three dimensional block of threads, and it provides a natural way to invoke computation across the elements in a domain such as a vector, matrix, or volume [68]. Each block (with all the threads in it) is assigned to a single SM, and blocks are independent from each other. Each thread within a thread block can: synchronize (one can specify synchronization points in the kernel by calling the __syncthreads() intrinsic function), share data and communicate efficiently using the shared memory. The number of threads in a thread block is chosen based on the amount of available shared memory and the memory access latency. However, this number is limited by the constrained memory resources of the processor core on which the block

resides. More specifically, for current GPUs, a thread block may contain up to 1024 threads [68].

Thread blocks of the same thread dimensionality are organized into a one-dimensional, two-dimensional, or three-dimensional grid. Figure 3.10 [68] demonstrates the relationship between threads, blocks, and a grid. Therefore, the total number of threads is equal to the number of threads per block times the number of blocks. The dimension of a grid is generally dictated by the size of the data being processed or the number of processors of GPUs, which it can greatly exceed. However, thread blocks within a grid are required to execute independently, and they can be executed in any order.

In addition, CUDA manages and executes threads in groups of 32 called warps, where threads are woven together and get executed in lockstep (CUDA 9 is able to use flexible synchronization) [75]. Although all threads in a warp execute the same instruction at the same time, each thread has its own instruction address counter and register state, and executes on its own data.

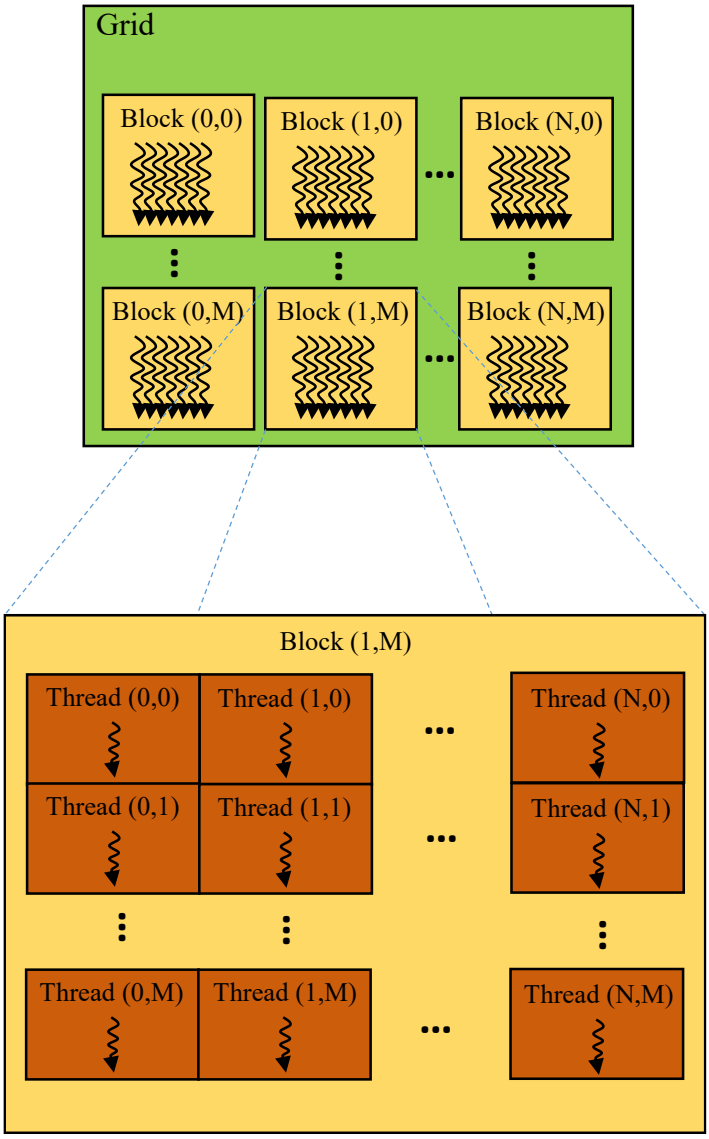

Figure 3.10 - The relationship between threads, blocks, and a grid. The total number of threads is equal to the number of threads per block time the number of blocks. The dimension of a grid is generally dictated by the size of the data being processed or the number of processors of GPUs, which it can greatly exceed.

3.4.2.5 Memory Hierarchy

Memory access and management are important parts of any programming language. CUDA threads may access data from multiple memory spaces that can be used to achieve high compute to global memory access (CGMA) ratios during their execution. There are various types or levels of memory available on the GPU device, each with distinct characteristics. Figure 3.11 illustrates an overview of the CUDA device memory model.

Each thread has access to private local memory and register. Registers are the fastest memory space on a GPU. The local memory is used for data that does not fit into registers. It is part of global memory, so it is uncached and much slower than registers. Shared memory is one of the key components of the GPU. Shared memory is allocated to thread blocks, and it is visible to all threads of the block and with the same lifetime as the block. Shared memory is an efficient means for threads to cooperate, facilitates reuse of on-chip data, and efficient usage of shared memory can greatly reduce the global memory bandwidth needed by kernels. Also, all threads have access to the same global memory, which is located off-chip on the main GDDR memory. Global memory, which is most commonly used memory on a GPU, has the largest capacity, but it has the highest latency.

There are also two additional read-only memory spaces accessible by all threads in the same location as the global memory: the constant and texture memory. The global,

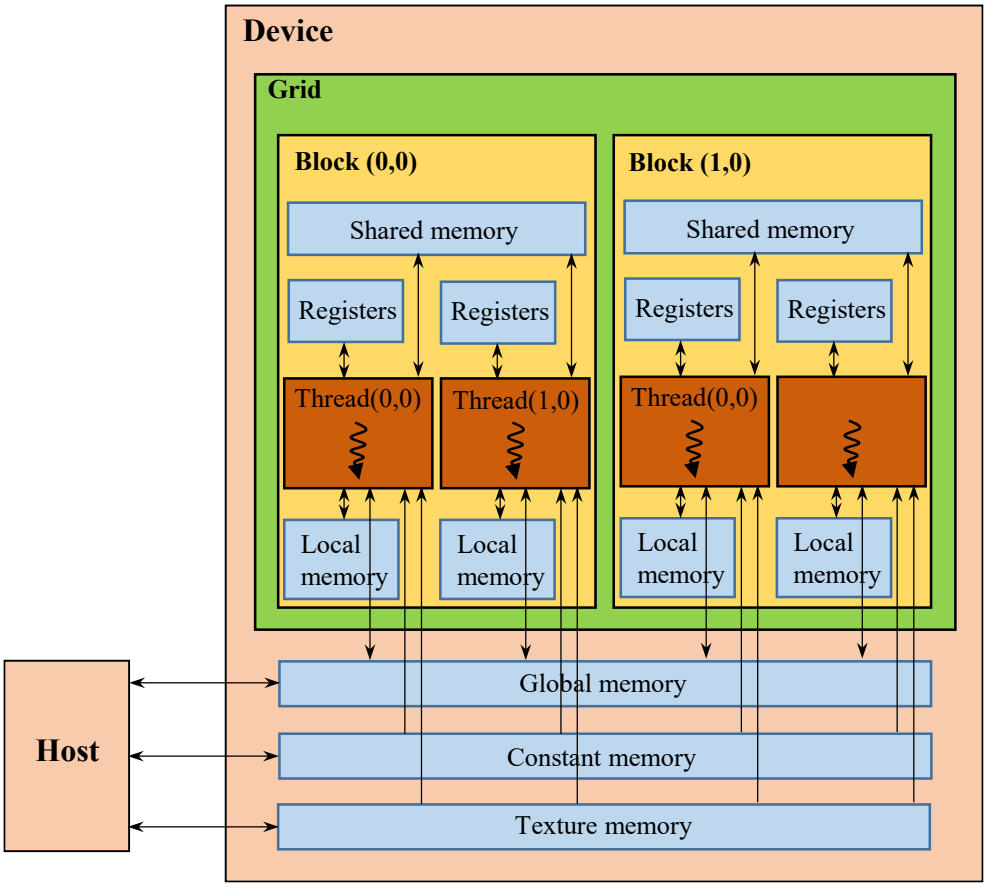

Figure 3.11 - Overview of the CUDA device memory model (Reproduced from [68]). Each thread has access to private local memory and register. Registers are the fastest memory space on a GPU. The local memory is used for data that does not fit into registers.

constant, and texture memory spaces are optimized for different memory usages. Constant memory resides in the same location as global memory and is cached. Threads can only read from constant memory, therefore it must be initialized by the host. Constant memory performs best when all threads in a warp read from the same memory address. Like constant memory, the read-only texture memory also resides in the same location as global memory, and is cached on chip. Specifically, it is optimized for 2D spatial locality, so best performance can be achieved when threads of the same warp read texture addresses that are close together in 2D [68].

Table 3.1 summarizes CUDA syntax for declaring program variables and their corresponding memory types [67, 76]. Table 3.1 also gives the scope and lifetime of each

memory. Scope specifies the range of threads that can access the variable, e.g., if the scope of a variable is a single thread, each thread will have the variable that can be only operated by the corresponding thread. Lifetime or lifespan, identifies the portion of the program's execution duration when the variable is available for use. For instance, shared memory has the same lifetime as the block, while constant memory variables exist for the lifespan of the application.

Table 3.1 CUDA variable type qualifier.

Qualifier	Memory	Scope	Lifespan
Automatic variables and fixed length, small array	Register	Thread	Thread
Automatic array variables	Local	Thread	Thread
__shared__	Shared	Block	Block
texture	Texture	Global	Application
__device__	Global	Global	Application
__constant__	Constant	Global	Application

3.4.2.6 Heterogeneous Programming

CUDA programming involves executing code on a host system and CUDA-enabled GPU devices. Figure 3.12 [68] demonstrates a heterogeneous programming model, where the CUDA kernels run on the GPU device and the rest of the C program executes on a CPU.

GPU devices are based on a distinctly different design from the host system, and the main differences are in threading model and separate physical memories. More specifically, host systems support a limited number of concurrent threads, whereas GPUs can support thousands of active threads concurrently. Threads on a CPU are generally heavyweight entities, but threads on GPUs are extremely lightweight. The host system and the device maintain their own distinct attached physical memories (host memory and device memory respectively).

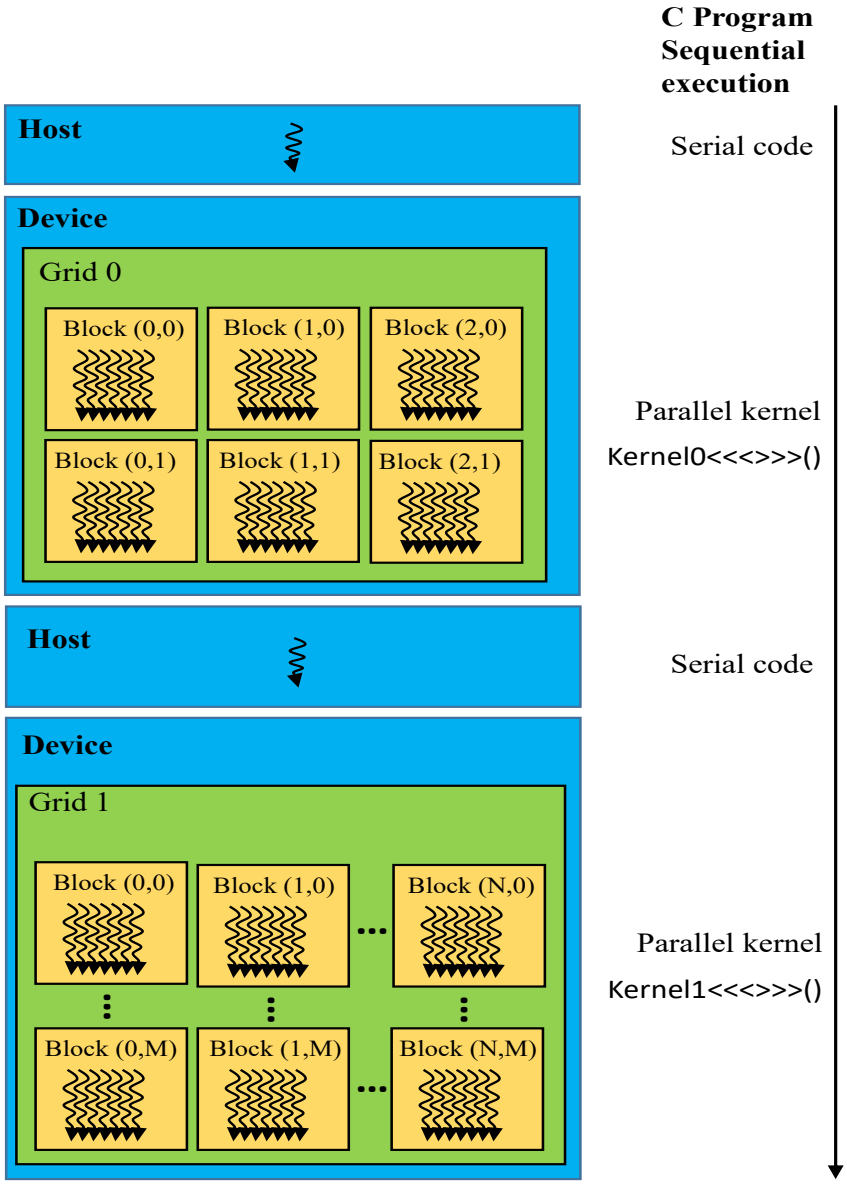

Figure 3.12 - Heterogeneous Programming model (Reproduced from [68]). The CUDA kernels run on the GPU device and the rest of the C program executes on a CPU.

3.5 GPU applications

GPU computing is increasingly being called upon for demanding consumer applications and high-performance computing. So far, there are over four hundreds of applications that are accelerated by GPU computing, and the number is growing [77]. Such applications fall into a variety of problem domains, including computational finance [78], climate & weather modelling [79], data science & analytics [80], deep learning and machine learning [81], medical imaging [82], and safety & security [83]. Compared with CPU only computing, GPU computing has achieved from 10× to more than 130× speedup [84].

3.6 GPU acceleration of standard FLIM algorithms

Before general-purpose GPUs were introduced, all software ran on CPUs serially and the GPUs were solely used for display. GP-GPUs have unleashed the power of GPUs for parallel computing. Due to high throughput floating-point operations and memory bandwidth, their applications have increased dramatically. There are many computational tasks that are well-suited for GPU processing. This is particularly true for applications that operate on a pixel-by-pixel basis, where each pixel is independent. FLIM analysis is one of the many applications that can be highly benefited from parallel computing with GPUs. Traditional FLIM systems based on CPU-only processing can hardly meet the demand of high-speed or real-time FLIM analysis.

As demonstrated in figure 3.13, a FLIM image consists of $n \times n$ independent pixels. The colour of each pixel is determined by the lifetime of the corresponding FLIM histogram. One can learn from this figure that FLIM analysis is highly suitable for parallel computing. In order to speedup FLIM analysis, it is necessary to calculate the lifetime of each pixel in parallel. GPUs, which have many parallel cores, are able to fulfil this requirement successfully.

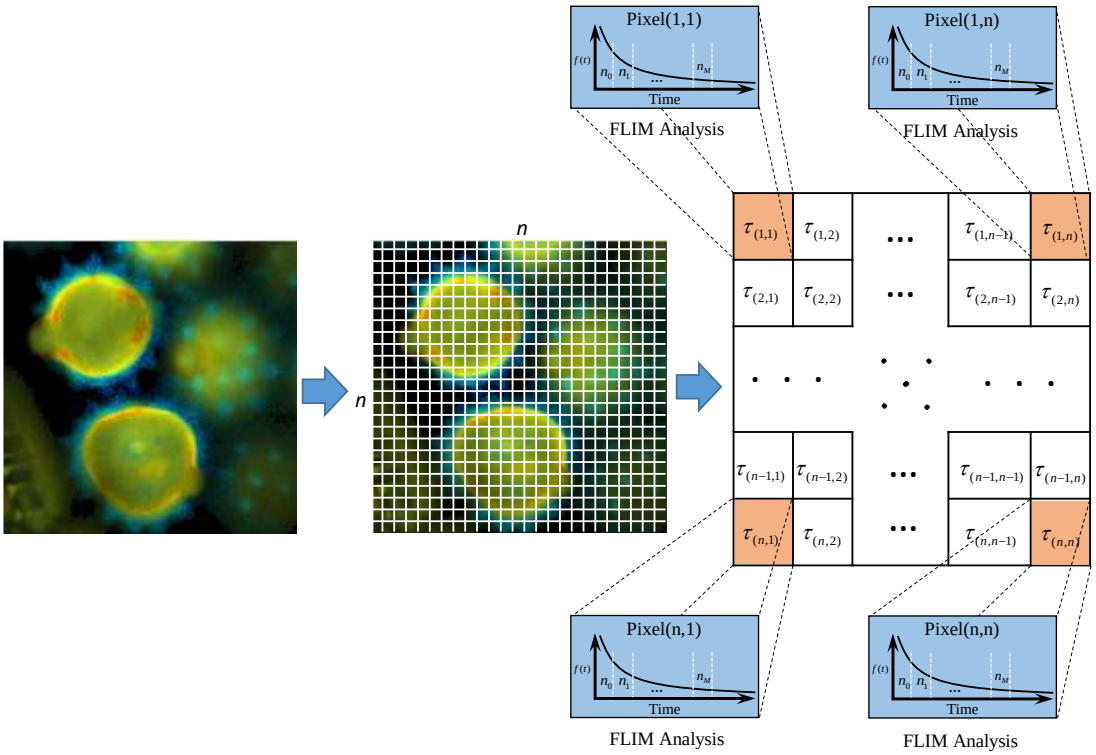

Figure 3.13 - The deconstruction of FLIM image. A FLIM image is consist of n×n number of independent pixels. The colour of each pixel is determined by the lifetime of corresponding FLIM histogram. One can learn from this figure that FLIM analysis is highly suitable for parallel computing. In order to speedup FLIM analysis, it is necessary to calculate the lifetime of each pixel.

GPU acceleration can be realised for any FLIM algorithm, and it can significantly speed up lifetime calculations compare to CPU-OpenMP (parallel computing with multiple CPU cores) or other CPU-only parallel computing techniques. Depending on the structure of each FLIM algorithm, different strategies can be applied for GPU implementations in order to achieve the best speed performance.

3.6.1 Thread-based computing for non-iterative algorithms

For non-iterative algorithms described in chapter 2, it is easy to implement GPU accelerations. GPU and CPU programs share similar code, except for the specific configurations of hardware resources and memory accesses in GPU implementations. The histogram of each pixel is analyzed by an independent CUDA thread, as shown in figure

3.14, and each block contains 512 threads [85]. This implementation strategy is designed based on the simplicity of the algorithms, where simple arithmetic operations plays the major role during lifetime calculations. Moreover, for each thread only a few resources (e.g., memory) are required and GPU devices are able to provide sufficient resources for all the threads that are running concurrently. This configuration allows analyzing a large number of pixels simultaneously, the exact number being determined by the number of streaming multi-processors (SMP). In this case, up to 30720 pixels can be launched on the NVIDIA Tesla K40 GPU [86]. Then, following the Single Instruction Multiple Threads (SIMT) mechanics [87], the lifetime estimations can be realized in parallel, instead of calculating lifetimes in a serial pixel-by-pixel manner on a CPU.

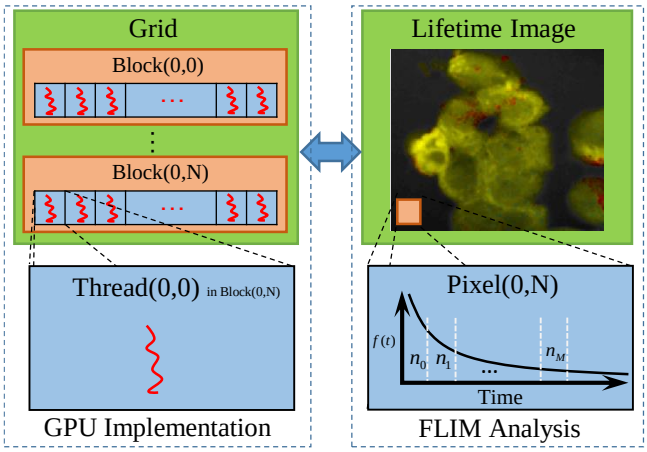

Figure 3.14 - The relationship between Thread, Block and Grid, and GPU implementation of non-iterative algorithms for FLIM analysis (Reproduced from [85]). Each thread processes an entire pixel.

3.6.2 Block-based computing for iterative algorithms

Different from simple algorithms, iterative algorithms require more complicated calculations and more computational resources are needed for lifetime calculation of each pixel. If the same implementation as simple algorithms were applied, huge resources demand (which exceeds what devices can supply) will limit the number of threads being executed at the same time. As a result, huge amount of computational units will be wasted. For iterative algorithms, instead, the histogram for each pixel is analyzed by a separate CUDA block, as shown in figure 3.15 (here, each thread processes a single time bin, instead of an entire pixel in non-iterative algorithms) and each such block contains 256

threads that correspond to the 256 time bins. This configuration still also allows for a large number of pixels to be analyzed simultaneously, and 120 pixels are expected on the NVIDIA Tesla K40. Here, we take LSM as an example to demonstrate how to implement iterative algorithms on GPUs. The CUDA operations can be divided into two categories:

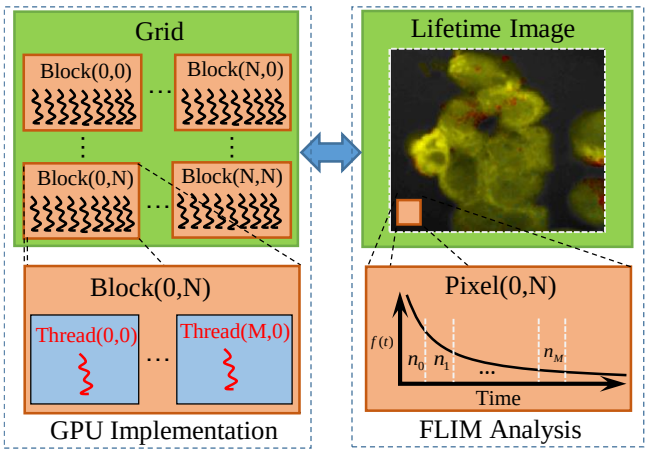

Figure 3.15 - The relationship between Thread, Block and Grid, and GPU implementation of iterative algorithms for FLIM analysis (Reproduced from [85]). Each thread processes on bin of the histogram of one pixel.

1) Thread-independent operation

For those operations which are independent in each time bin, the CUDA provides a basic instruction. For example, subtraction or division for each time bin $(N_j\text{-}Y_j)/\sigma_j$ can be finished in only one instruction cycle of one instruction as shown in figure 3.16(a) (here, we just take subtraction as an example, since division follows the same procedure), whereas there are 64 subtraction cycles in the CPU-OpenMP operation (figure 3.16 (b)). In figure 3.16 (a), N_j and Y_j are stored in the variables SubA and SubB, respectively, and every thread has these two variables, whereas the CPU defines two arrays (SubA[256] and SubB[256]) for loop operations.

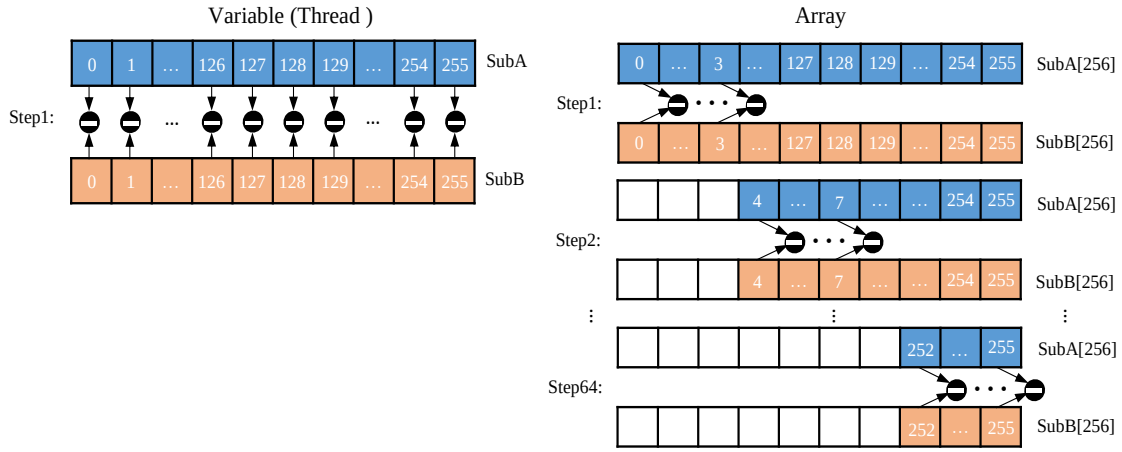

Figure 3.16 - The subtraction in GPU and CPU-OpenMP processing (Reproduced from [85]). Subtraction can be finished in only one instruction cycle of one instruction in GPU, whereas there are 64 subtraction cycles in the CPU-OpenMP operation.

2) Communication between threads

Alternatively, the CUDA provides a special solution for some operations where communication or data sharing with other time bins (threads) is necessary. As shown in figure 3.17(a)[75, 88], the data $(N_j\text{-}Y_j)^2/\sigma_j^2$ of each time bin is stored in the jth element of a shared memory array SumA (A shared memory is an on-chip high-speed memory, which enables the threads in the same block to access the same memory content. Arrays are defined by: __shared__ SumA [256]; in CUDA programming [86]). In Step 1, the first 256/2 = 128 threads compress the array SumA into 128 elements by combining the jth and (j+128)th units (j = 1,…,128) simultaneously, followed by several similar steps. Finally, after only $\log_2 256=8$ steps (it is not a requirement that the number of threads in a block has to be a power of 2, but it maximizes performance), the summation reduction result can be acquired, instead of after 65 steps in the CPU-OpenMP version as shown in figure 3.17(b).

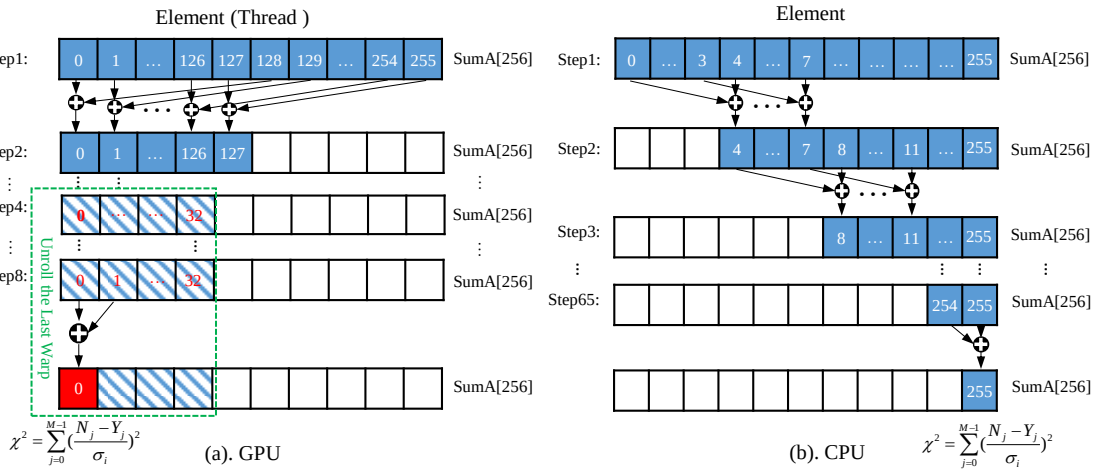

Figure 3.17 - The summation reduction in GPU and CPU processing (Reproduced from [85]). After only $\log_2 256=8$ steps, the summation reduction result can be acquired in GPU, instead of after 65 steps in the CPU-OpenMP version.

3.7 Optimization of GPU Programming

GPU acceleration performance can vary with utilization strategies. In order to maximize the power of GPU in FLIM analysis, it is inevitable to consider the specifications of GPU hardware, manage the memory properly, and design a specific programming strategy for each individual algorithm [85]. For standard simple algorithms, which applied with similar implementation as CPU programming, the kernel configuration and overlap of data transfer are the main factors that influence their performance. As for the rest of algorithms (including ANN-FLIM), the following considerations are all needed to deal with appropriately.

3.7.1 Kernel Configuration

The CUDA kernel is launched in a host (CPU), which is the entrance of GPU processing similar to the main function in the C language, a proper size of grid (determines the dimensions of blocks) and block (determines the dimensions of threads in a block) should be determined in order to optimize the GPU performance. On CUDA computing, every 32 threads are bounded together into a so-called warp, and it is executed in lockstep (this will be changed from CUDA 9). Accordingly, it is suggested that the number of time bins

of FLIM histogram data (this can be done by neglecting some bins that are less important for lifetime calculations) and the thread number within a block should be a multiple of 32, e.g., 256, as well as the number of neurons in each layer. Furthermore, according to the GPU hardware specification, at most 2048 threads or 16 blocks can be launched in a Streaming Multiprocessor (SM) (the number of SM varies with specific GPU hardware), which is the workhorse of the GPU[89] and puts a different constraint on the number of threads. Table 3.2 illustrates that for thread numbers, which are not multiples of 32, although fewer threads are launched, the total processing time is always longer. This is due to so-called warp divergence, which occurs if threads in the same warp follow different conditional branches in the code. In CUDA diverging branches in a warp are executed serially (only threads in the current branch are working, and others are waiting), and branch divergences therefore lead to considerably slower execution. Additionally, when the thread number is 224, although it is a multiple of 32, its operation time is longer than expected: 1662.5 > (224/256)×1870.2 = 1636.4ms. This is because for each SM only 2016 (< 2048) threads have been launched, so that not all the hardware resources are being used.

In this case, considering a 512x512 FLIM image to be generated, we examine the performance of GPU FLIM with a different grid size. As showed in Table 3.3, for FLIM analysis, the calculation times for different sizes are almost the same, so there is no need to configure the block dimension, as long as it follows the limitation (maximum grid size of NVIDIA K40: $<2^{31}$-1, 65535, 65535>[86]).

Although simple algorithms use the different configurations (i.e. different size of grid or block), the same features can be found as illustrated above.

Table 3.2 Operation time under different block size of LSM-GPU (Reproduced from [85]).

Threads per Block	**Warp divergence**	**Full launch**	**Time** (ms)	**Time/bin**(ms)
256	No	Yes	1870.2	7.305
255	Yes	No	1898.9	7.447
253	Yes	No	1912.8	7.560
250	Yes	No	1914.2	7.657
225	Yes	No	1960.4	8.713
224	No	No	1662.5	7.422

Table 3.3 Operation time under different grid size of LSM-GPU (Reproduced from [85]).

1st Dimension	**2nd DIMENSION**	**Time** (ms)
512*512	1	1869.4
512*16	32	1867.2
512*2	256	1871.3
512	512	1870.2
32	16*512	1872.6
2	256*512	1870.7

3.7.2 Memory Optimization

Memory usage and throughput are very important for GPU computing, and as they can highly influence the performance, it is necessary to optimize them appropriately. For FLIM analysis, the following three aspects should be considered.

First, for a real-time FLIM process, histogram data on CPU memory is updated frequently, and the GPU has to access the data from CPU for every image frame. So the mapped pinned (M-pinned) memory should be encouraged, which is the host memory and has been page-locked and mapped for direct access by the GPU, as shown in Fig. 3.18 [89]. M-pinned memory makes sure the data will be updated on the same address and the GPU does not need to look for a new address in order to get the data. Also, to

minimize data transfers between the CPU and GPU, the GPU FLIM analysis only uses device memory after the histogram data has been transferred from the host.

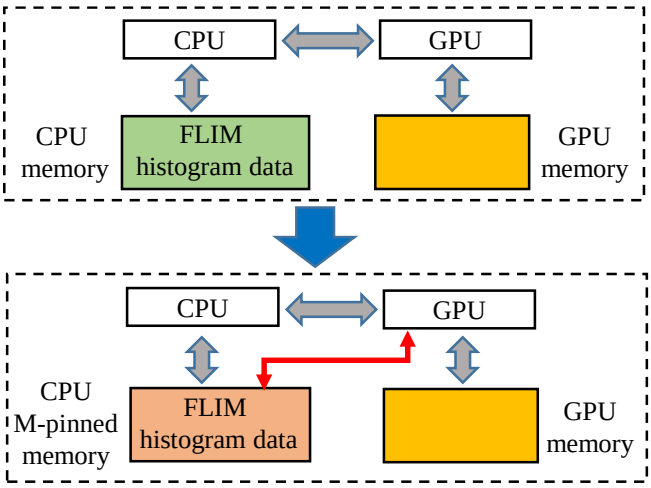

Figure 3.18 - GPU-FLIM analysis with mapped pinned memory (Reproduced from [85]). The mapped pinned memory of CPU has been page-locked and mapped for direct access by the GPU.

Moreover, when a large amount of data is used (e.g., large number of time bins, large image size), asynchronous and overlapping transfers with computations [90] have to be considered to improve the computation performance. CUDA streams (which are sequences of operations that are performed in order on the device) can be used to fulfill this mission. As shown in figure 3.19(a), in sequential mode, the kernel can only be launched after all the data has been transferred to the GPU memory. In this mode, only one type of operations can be executed at any time, e.g., lifetime calculation can be only started after the whole histogram data of an entire image have been transferred to GPU device. In contrast, for the concurrent mode, the pixels of a FLIM image are divided into several groups, and each pixel within the same group will be launched and processed in the same stream. As illustrated in figure 3.19(b), histogram or lifetime data transfer and calculation kernels are separated into different streams. This configuration allows data transfer and kernel execution to run simultaneously, and can reduce the whole processing time measurably. In this case, three streams are launched, and the overall processing time is less than the sequential model, as demonstrated in the red dash line. For real-time FLIM analysis, the concurrent execution mode is even more important, as there is new data coming all the time.

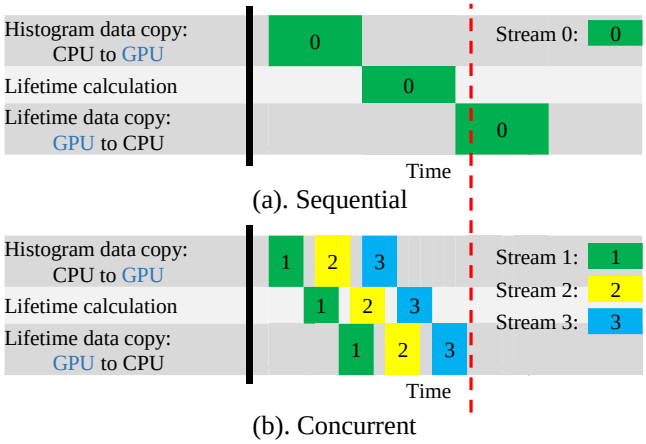

Figure 3.19 - Timeline comparison of sequential and concurrent executions (Reproduced from [85]). The sequential execution has only one stream, however, the concurrent execution has two or more streams.

Last but not least, as discussed in chapter 3, GPUs have a number of memory spaces including the shared, global, constant, texture, register and local memory [86, 90], and for GPU performance, one of the most important areas is memory management [90]. Among these memories, the shared memory is on-chip memory, and it has much higher bandwidth and much lower latency than the local or global memory. It is highly recommended that the shared memory should be used as much as possible, especially when there is communication between threads in the same block (e.g., summation reduction operation). However, shared memory is very limited in size and it is unavoidable to use the large global memory. Because global memory is by far the slowest memory accessible from the GPU, care should be taken to access it in the most efficient way. CUDA supports so-called coalesced memory access, where several threads of a warp access a congruent area of device memory in a single memory fetch. Using coalesced access as much as possible is very important to minimize the necessary bandwidth. Figure 3.20 illustrates the difference between the coalesced access and non-coalesced access, and the ensuing large speed difference [67, 86]. More specifically, this implies, for example, that for non-iterative algorithms, the histogram data needs to be arranged such that the data of the same time bin, but different pixels are directly adjacent, since each thread will access data of the same time bin of the corresponding pixel. For traditional iterative and ANN-FLIM algorithms, however, data of each time bin of a pixel has to be stored in a continuous memory locations. Table 3.4 demonstrates that coalesced

access is more than 7-fold faster than non-coalesced access for the data of a 512×512 size image [85]. In addition, the constant and texture (optimized for 2D spatial locality) memory space, which is cached and read only, can be used to save constant arrays or matrices (e.g., weight matrices of ANNs) used in the algorithm, and can also speed up the processing. For example, for IEM, every pixel uses the same coefficients C_j and we can calculate the value before processing, so C_j can be kept in the constant memory (declared by "__constant__ float CoIEM[]").

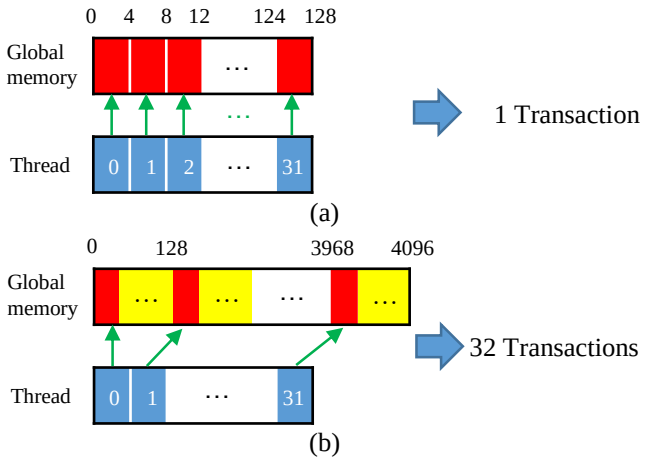

Figure 3.20 - Global memory access: (a) coalesced access; (b) non-coalesced access (Reproduced from [85]). Using coalesced access as much as possible is very important to minimize the necessary bandwidth.

Table 3.4 Comparison between coalesced access and non-coalesced access of global memory.

Access mode	**Image size**	**Bin Numbers**	**Data volume** (mb)	**Time** (ms)	**Speedup**
Coalesced	512×512	256	6109.6	1.8	7.2
Non-coalesced				13	

3.7.3 Programming Strategies

The first consideration is to trade the precision for the speed when it does not affect the result, such as using single-precision operations instead of double-precision ones. As for FLIM analysis, single-precision operations for lifetime calculations is able to provide sufficient results, and this trade off can significantly improve the speed performance.

Then, the number of divergent warps (threads in the same warp have different operations) that cause delay have to be minimized. Figure 3.21 shows a simple example of how to avoid divergence. It demonstrates that shifting elements by changing the index can be used to reduce the divergence in some circumstances. Here, the original operations across two warps, where four operation branches are created. By changing the index, all operations can be done in one warp and only one branch is generated. More importantly, it is important to reduce the number of waiting threads. For example, we execute two summation reductions together in different directions as shown in figure 3.22. If we consider only one summation reduction, last 128 threads become idle after one step of operation. Instead, we run another reduction in an opposite direction (as shown in the right of figure 3.22), and all the threads will be occupied in the second step. As a result, two groups of operations are running simultaneously, and the number of idle threads is reduced as shown in figure 3.22.

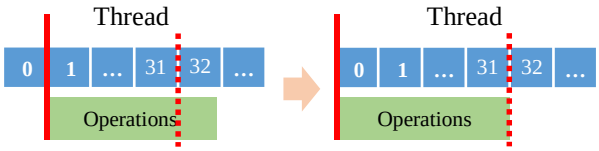

Figure 3.21 - Example of avoiding warp divergence (Reproduced from [85]). In some circumstances, shifting elements by changing the index can be used to reduce the divergence.

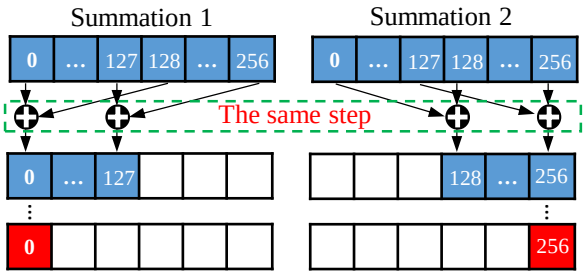

Figure 3.22 - Example of maximizing hardware usage (Reproduced from [85]). In this case, two groups of operations are running simultaneously, and the number of idle threads is reduced.

Moreover, since GPU devices have been widely used, there are a large amount of useful resources that can be found, e.g., programming strategies for certain problems and CUDA tools. More specifically, CUDA provides some libraries (e.g., cuBLAS) and GPU versions of math functions, and using these resources can not only boost the performance

but also can reduce the difficulty of CUDA programming, e.g., using "__fdividef()" for floating point division.

3.8 Evaluation of standard algorithms

To demonstrate the advantage of the proposed GPU accelerated FLIM analysis tool, algorithms were tested on both synthesized and experimental data, and the results were compared with CPU solutions. Results were compared in terms of the performances, i.e. precision in FLIM analysis, and the time of processing, i.e. their speed. The results are based on a NVIDIA TESLA K40 GPU and an OpenMP CPU implementation on an Intel(R) Xeon(R) E5-2609 v2 processor with 4 cores.

Most of the simple algorithms are designed for single-exponential decay analysis. Here, we assume the theoretical FLIM image is a square with 4 color bars (each color represents a lifetime). More specifically, as shown in figure 3.23(a), the lifetime gradually increases from left to right, and the colorbar on the right of figure 3.22(b) defines the mapping of color to lifetime. In this case, the lifetimes of the four bars are 2, 2.5, 3, 4ns, respectively, which means every pixel within the same bar has the same theoretical lifetime.

The FLIM image contains 512 by 512 pixels, and each histogram has 256 bins with the bin width of (100ps). In order to make a fair comparison, the same FLIM data is used for all the algorithms to generate FLIM images, and each pixel in the same color bar has similar number of photons (around 2000). Both CPU and GPU computations use single-precision operations, and have identical outputs, although they have different architectures, i.e. latency oriented vs. throughput oriented. As shown in figures 3.23 (b)-(d), one can easily find that all algorithms generate effective results in agreement with the known ground truth, although IEM is not as good as others. From the results, we can also learn that the averaged relative precision values (the average value of σ_τ/τ) of IEM, CMM, and LSM are 0.15, 0.026, and 0.03. These results further confirm that, in terms of precision, CMM and LSM are better than IEM. Generally, iterative algorithms provide similar or better results, compared with simple algorithms.

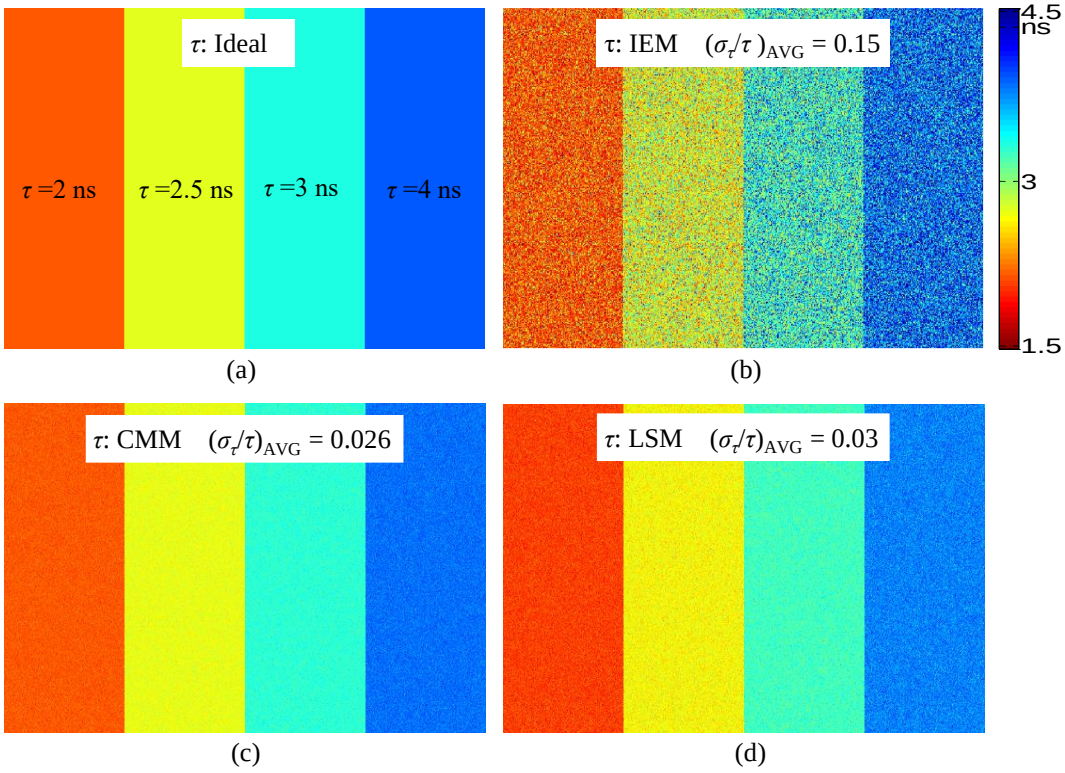

Figure 3.23 - FLIM images of single-exponential decays: (a) theoretical lifetime image; (b) IEM; (c) CMM; (d) LSM (Reproduced from [85]). The theoretical FLIM image is a square with 4 color bars. All algorithms generate effective results in agreement with the known ground truth, although IEM is not as good as others. The averaged relative precision values (the average value of σ_τ/τ) of IEM, CMM, and LSM are 0.15, 0.026, and 0.03.

For bi-exponential decays ($y(t)=Af_D\exp(-t/\tau_F) + A(1-f_D)\exp(-t/\tau_D)$), there are three FLIM parameters, i.e., two lifetimes (τ_D, τ_F), and f_D. In this simulation, instead of changing lifetime values across color bars, every pixel has the same lifetimes (τ_D = 3ns, τ_F = 1 ns), whereas f_D decreases from 0.8 to 0.2 from left to right of the image. Table 3.5 provides the information of simulation setup, e.g., image size, number of bins, bin width. The average lifetimes ($\tau_{AVG} = f_D \times \tau_F + (1-f_D) \times \tau_D$) calculated by different algorithms are given in figures 3.24(b)-(d), where 32 means a segment of GA contains 32 pixels [39-41], compared with the theoretical image as in figure 3.24(a). From these images, we can learn that all the algorithms generate satisfactory results when $f_D > 0.5$, whereas the GA shows the greater potential as it can resolve a wider range of f_D. The averaged relative precision values of each algorithm confirm that GA has much better overall performance.

Table 3.5 Details of bi-exponential simulations

Image Size	**Bin Number**	**Bin Width** (ps)	τ_D (ns)	τ_F (ns)
512x512	256	100	3	1

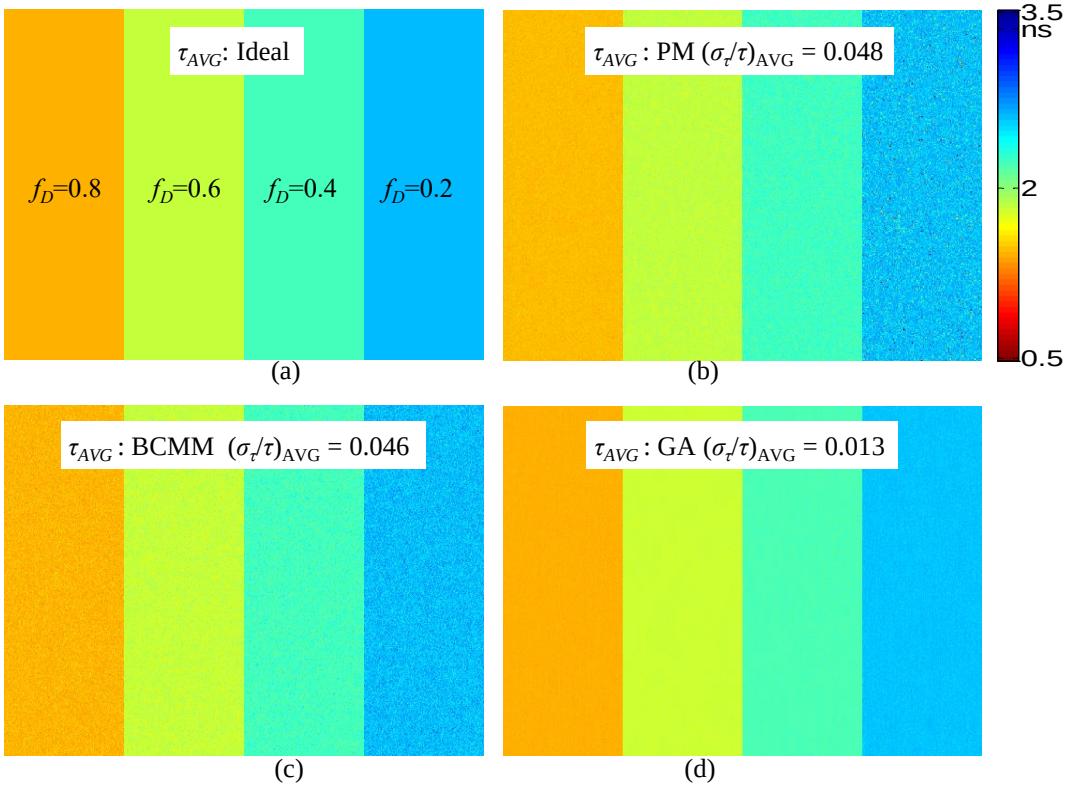

Figure 3.24 - Images based on synthesized bi-exponential data: (a) theoretical average lifetime image; (b) PM; (c) BCMM; (d) GA-32 (Reproduced from [85]). All the algorithms generate satisfactory results when $f_D > 0.5$, whereas the GA shows the great potential as it can resolve a wider range of f_D. The averaged relative precision values of PM, BCMM, and GA are 0.048, 0.046, and 0.013.

Tables 3.6 and 3.7 show the processing time and speed enhancement of the GPU implementation against the parallel OpenMP implementation on the CPU. In Tables 3.6 and 3.7, we also included the acquisition rates of recently reported FLIM systems [20, 34] for comparison. These systems are all for 256x256 images, but we extended them for 512x512 ones in order to make a proper comparison. For the latest FLIM systems, the acquisition can range from 0.02 to 2.5 fps (note that the frame rate reported in [18] will be much slower if more gates are used as suggested in their experimental section). It shows that the acquisition would be the bottleneck if simple algorithms are applied, unless wide-field acquisition is applied [35]. Tables 3.6 and 3.7 also show that GPUs do offer speed enhancement when a more precise analysis (GA-32 or LSM) is needed, and the

analysis frame rate is comparable to the acquisition rate. Due to different structure and processing schemes, the enhancement factor for each algorithm is different. Table 3.8 gives a deeper understanding about each algorithm implemented in GPU by using the CUDA profiler, where one can see the time of data transfer between GPU and CPU, achieved GPU occupancy, etc. From these results, one can find that iterative algorithms have much lower occupancy. This is due to the use of conditional and flow control statements and higher warp divergence, where more idle threads are created. Also, the acquisition (using the system reported in [34] as a reference) and image analysis with different image resolutions are shown in figures 3.25-3.27 for both simple and iterative algorithms. Since algorithms in the same category share the similar features, here, we only take IEM, LSM and GA as examples. Figure 3.25 illustrates the acquisition time and GPU image analysis time for IEM with different image resolutions. Compared to the acquisition time, the analysis time is negligible. The acquisition time and GPU image analysis time for LSM with different image resolutions have been presented in figure 3.26. For LSM, the analysis time is also less than the acquisition time. Whereas, as demonstrated in figure 3.27, due to the complexity of GA, the analysis time is around 2-fold longer than the acquisition time. In figure 3.28, the ratio of GPU image analysis time and acquisition time for three FLIM algorithms with different image resolutions is illustrated. For IEM and other simple algorithms, the ratio is rather stable across image resolutions. Since the acquisition time falls proportionally to the number of pixels, the analysis time also is in proportion to the number of pixels in a consistent manner. However, as for the iterative algorithms, the ratio grows when the image size decreases, and this means the analysis times are no longer in proportion to the number of pixels. The more complicated the algorithm is the faster the ratio grows. These results also indicates that simple algorithms are more likely to exploit the power of the GPUs.

Table 3.6 Processing time for single-exponential decay (unknown: τ, K) (Reproduced from [85])

Algorithm	**CPU** (ms)	**GPU** (ms)	**Speedup** (times)	**Acquisition Rates** (fps)	**Image Analysis Frame Rate** (fps)
IEM	355.2	22.4	15.9	0.02~2.5	44
CMM	370.3	23.1	16.0		43
LSM	21463.1	1170.2	18.3		0.85

Table 3.7 Processing time for double-exponential decay (unknown: τ_F, K, f_D) (Reproduced from [85])

Target	**Algorithm**	**CPU** (ms)	**GPU** (ms)	**Speedup** (times)	**Acquisition Rates** (fps)	**Image Analysis Frame Rate** (fps)
τ_F	PM	472.2	22.8	20.7	0.02~2.5	44
	BCMM	537.1	26.3	20.4		38
	GA-32	76410	3313.8	23.1		0.3
τ_{AVG}	PM	520	22.9	22.7	0.02~2.5	44
	BCMM	451.4	23.3	19.4		43
	GA-32	77360.7	3320.2	23.3		0.3

Table 3.8 Results from CUDA profiler for each algorithm (Reproduced from [85])

Algorithm	**Data Transfer (ms)**	**GPU Computation (ms)**	**Occupancy**	**Warp Efficiency**
IEM	18.7	3.7	99.0%	99.8%
CMM		4.7	96.0%	96.0%
PM		4.1	99.0%	100.0%
BCMM		7.6	99.2%	100.0%
LSM		1151.5	54.1%	97.8%
GA		3295.1	52.9%	98.0%

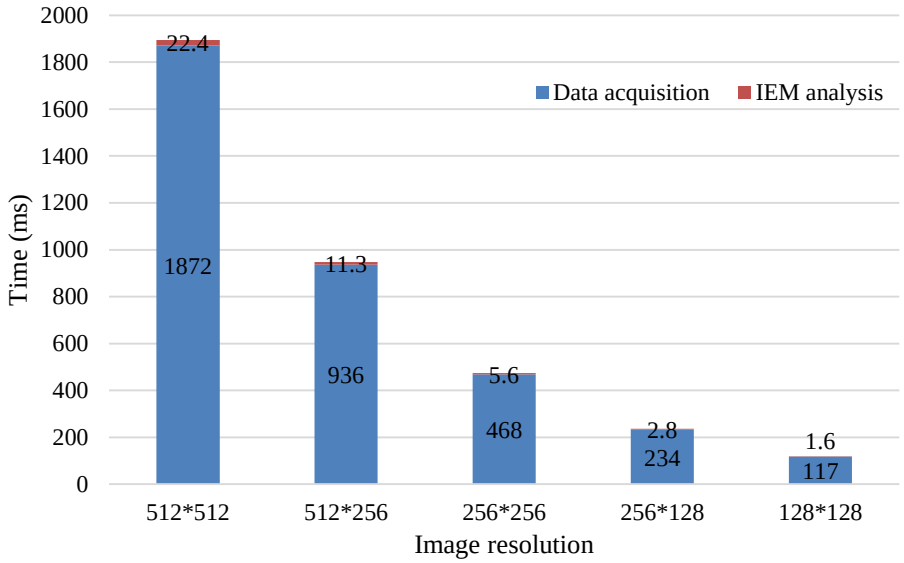

Figure 3.25 - Acquisition time and GPU image analysis time for IEM with different image resolutions (Reproduced from [85]). Compared to the acquisition time, the analysis time is negligible.

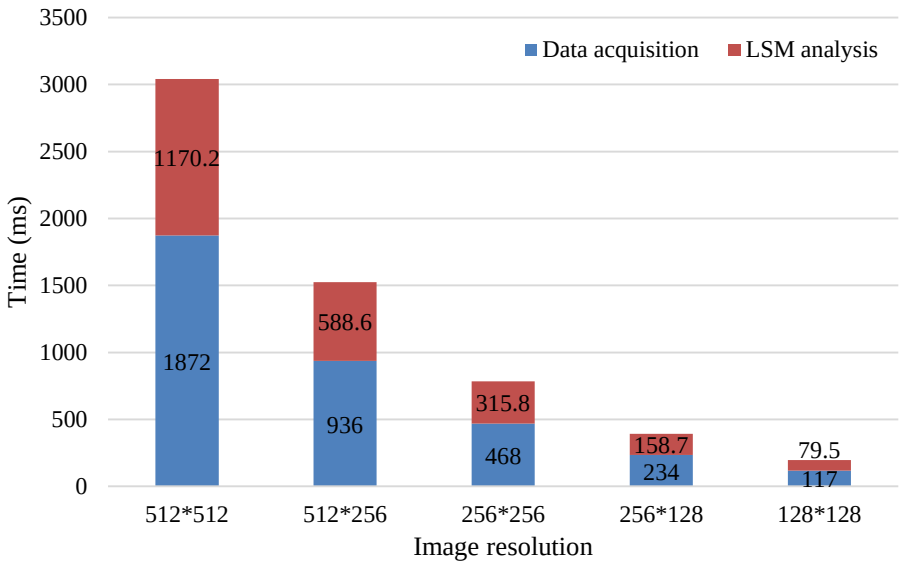

Figure 3.26 - Acquisition time and GPU image analysis time for LSM with different image resolutions (Reproduced from [85]). For LSM, the analysis time is also less than the acquisition time.

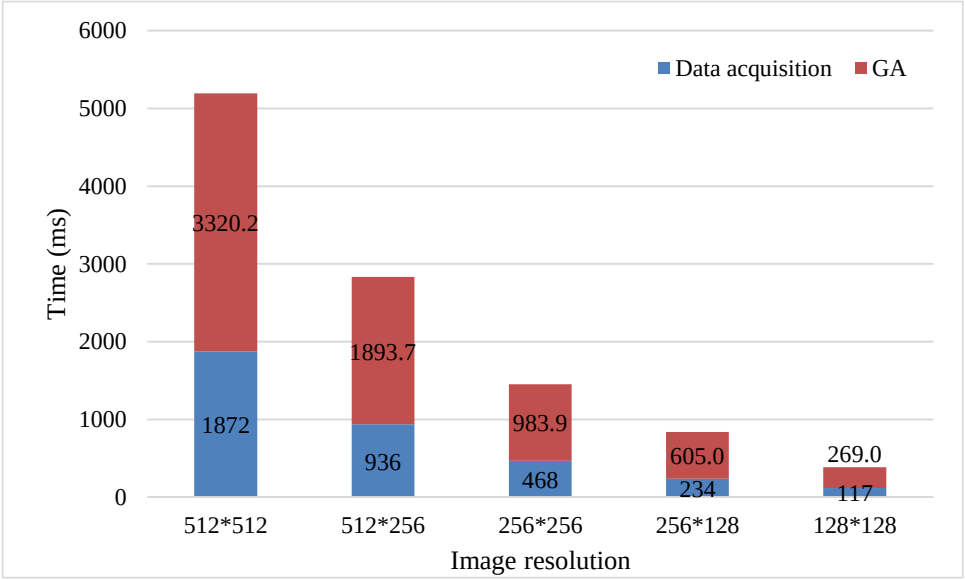

Figure 3.27 - Acquisition time and GPU image analysis time for GA with different image resolutions (Reproduced from [85]). Due to the complexity of GA, the analysis time is around 2-fold longer than the acquisition time.

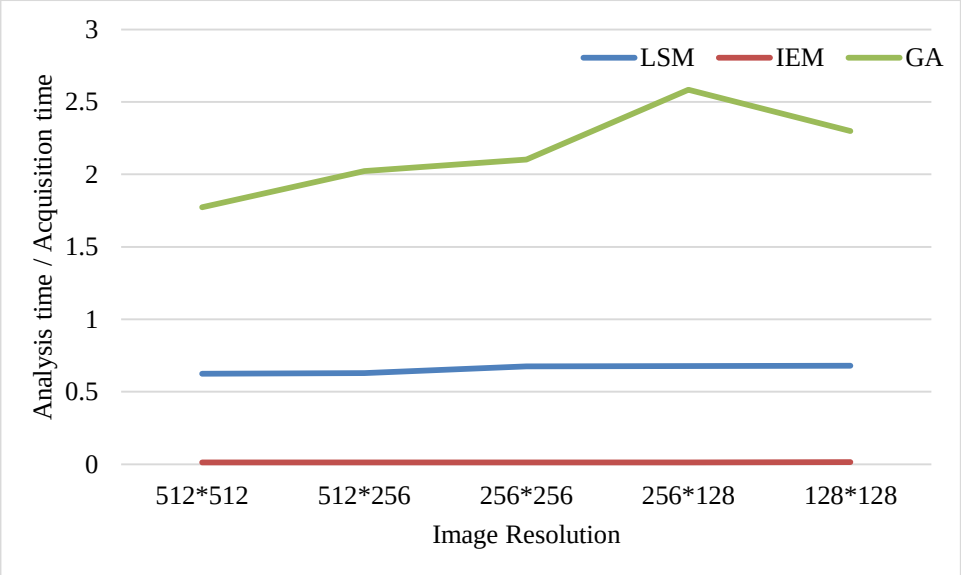

Figure 3.28 - The ratio of GPU image analysis time and acquisition time for three FLIM algorithms with different image resolutions. For IEM and other simple algorithms, the ratio is rather stable across image resolutions. Since the acquisition time falls proportionally to the number of pixels, the analysis time also is in proportion to the number of pixels in a consistent manner. However, as for the iterative algorithms, the ratio grows when the image size decreases, and this means the analysis times are no longer in proportion to the number of pixels. The more complicated the algorithm is the faster the ratio grows.

3.9 Summary

This chapter briefly describes the reason why parallel computing is necessary for FLIM analysis. Also, the GPU architecture is demonstrated, where one can learn that GPUs contain thousands of parallel cores. With the advent of CUDA, developers are able to explore the great potential of GPUs for general purpose parallel computing applications. As one of the most popular parallel processors, GPUs have been successfully used in hundreds of applications for both academic research and industry. Also, standard FLIM algorithms have been implemented on GPUs with corresponding strategies. The results shows that GPU accelerated standard algorithms are able to achieve at least 15x speedup, compared with CPU-only systems.

Chapter 4 Artificial neural network and FLIM analysis

4.1 Overview

Accurate FLIM parameters estimation is a very complicated task due to the existence of noise, especially when the total photon count is low. In fact, it is very challenging to acquire a large number of photons for each pixel, and the main reasons are: 1) most samples will only emit a limited number of fluorescence photons before the onset of photobleaching or photodamage [91]; 2) more photons means longer acquisition time, and it will be more difficult to build a high-speed FLIM system. The IRF makes lifetime analysis even more complicated. Therefore, it is critical to develop an algorithm that can work on noisy data from a few photons and deliver faster FLIM analysis.

Artificial neural networks (ANNs), inspired by biological neural networks of the brain, have been applied in many fields. ANNs are known to be able to approximate continuous function. This allows the use of ANN techniques for FLIM analysis as the lifetime parameters can be interpreted as functions of the lifetime histograms. This chapter demonstrates that without any time-consuming iterative processes, a pre-trained feedforward ANN is capable of calculating FLIM parameters quickly. This chapter provides a brief overview of ANN, as well as some of its applications. Then the details of how to design and use the ANN based FLIM algorithm (ANN-FLIM) will be demonstrated.

4.2 Artificial neural networks

Machine learning (ML) that automatically learns programs from data has become more and more popular for data analysis [92]. Artificial neural networks (ANNs), inspired by biological neural networks, are one widely used class of ML algorithms and artificial intelligence techniques[93, 94]. It is an advanced interpolation method. ANNs represent human's feeble attempts to replicate thinking or decision making processes in the brain (e.g., memory, recognition, prediction, and planning). ANNs are relatively crude electronic network models which try to mimic the neural structure of the human brain. The neurons in the brain responds to the new external stimuli based on previous experiences. It is natural proof that small energy efficient packages (connected neurons) are capable of solving some problems that are beyond the scope of current computers [95, 96]. Although ANNs have limited functionality compared with the human brain, this new method of problem solving provides a more graceful degradation during system overload than traditional methods. ANNs and other artificial intelligence approaches of computing are thought to be the next major advancement in the computing industry [95, 97].

The concept of the ANN analysis was developed more than 70 years ago. In 1943, the first developments in ANNs began with the invention of a computational model for neural networks based on mathematics and algorithms called threshold logic by Warren McCulloch and Walter Pitts [98]. In the late 1940s, psychologist Donald Hebb developed a rule for training based on the mechanism of neural plasticity [99], which was later used to derive most of the modern training rules for ANNs. Frank Rosenblatt (1958) created the perceptron which is one of the major contributions for ANNs. The perceptron is an algorithm for supervised learning of binary classifiers. In 1974, Paul Werbos proposed the backpropagation algorithm [100], which effectively solved the major training problem. Recently, thanks to the development of high-performance computing technologies, ANN applications have dramatically increased in as a key solution in many fields.

ANNs are capable of solving highly complex and time-consuming computational problems. It is also known that ANNs are more favourable for problems that have

statistical inputs (e.g., load forecasting and fault location). Operated like a black-box model, ANNs do not require detailed information about the system, which is usually difficult to acquire. Once trained properly, they can effectively learn the relationship between the inputs and the outputs, and are able to generate reliable results when given new inputs. What's more, ANNs have the ability to solve highly nonlinear problems, and can work with both numerical and analogue data. ANNs are relatively robust and less affected by noisy or inconsistent data, and it is not necessary to have sophisticated mathematical knowledge in order to use them.

4.2.1 A biological neuron

A neuron is the fundamental processing element of a neural network, and is an electrically excitable cell that processes and transmits information through electrical and chemical signals. A neuron combines received inputs, then generates outputs by performing a nonlinear operation. Figure 4.1 demonstrates the structure of a biological neuron.

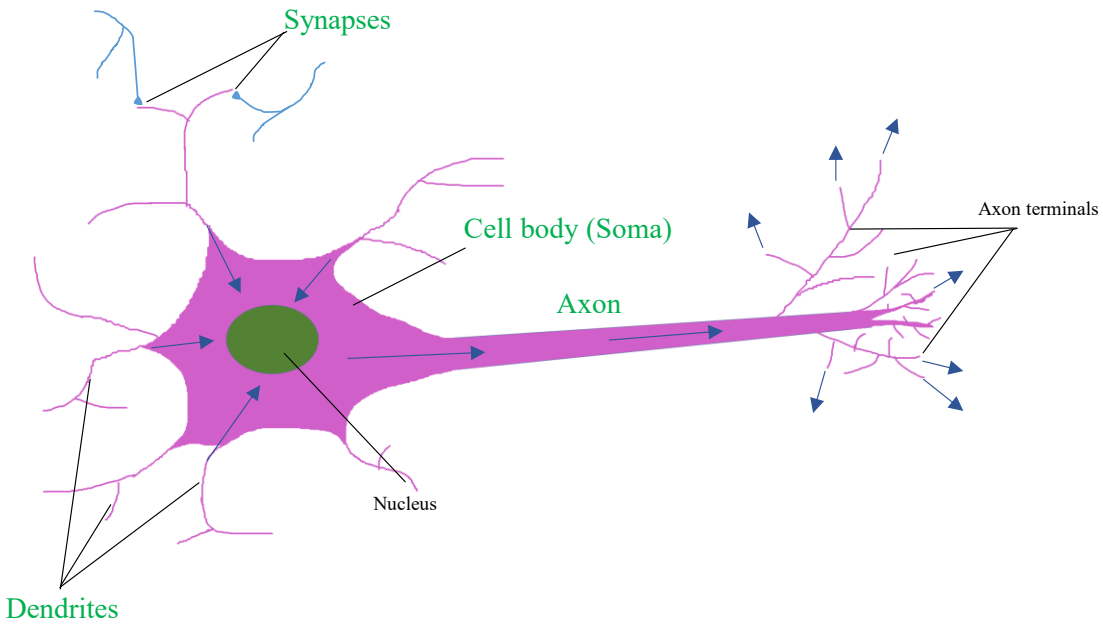

Figure 4.1 - Structure of a biological neuron. The biological neurons consist of four basic components: a cell body (soma), dendrites, an axon, and synapses.

Although there are various types of neurons, all the biological neurons consist of four basic components, i.e., a cell body (soma), dendrites, an axon, and synapses. A synapse is a structure that allows a neuron to pass an electrical or chemical signal to other neurons. Dendrites are tree-like extensions of the cell body which accept incoming electrical signals into the cell body. A cell body is the body of neuron cell containing the cell nucleus and effectively processing incoming signals. An axon is single long fibre (a long tubular structure) which sends out the outputs to other neurons. Figure 4.1 also illustrates the flow of electrical signals, i.e., from synapses to dendrites, cell body, and then axon.

4.2.2 Mathematical model of an artificial neuron

Similarly to the biological neuron, an artificial neuron is the constitutive unit of artificial neural networks. Although there are many neural network models that have been developed with different theory, most of them are made of similar artificial neurons with different interconnection structures. Figure 4.2 illustrates the most used model of a neuron, which was initially proposed by Warren McCulloch and Walter Pitts. Each neuron has several inputs and one output. It consists of weights summation (determines how the network inputs are combined inside a neuron) and activation function (determines the output). Then a weighted linear combination of inputs is given by

$$z = \sum_{i=0}^{n} w_i x_i \tag{4.1}$$

where the input signal x_i is multiplied by a corresponding weight w_i, which can either be a positive or negative value. x_0 is equal to 1, and w_0 is usually known as the bias b (or threshold). The output of the neuron is denoted by

$$y = f(z) \tag{4.2}$$

where f is the transfer function (or activation function), which computes the output of the neuron. The transfer functions are chosen to have various properties which either enhance or simplify the neural network containing the neuron, and the transfer function selection

plays a significant role in the neural network design. There are several types of transfer function, e.g., step function, linear function, log-sigmoid function, hyperbolic tangent function (or tan-sigmoid), radial basis functions [101], and rectified linear unit (ReLU) [102]. Among the transfer functions, linear, log-sigmoid, and tan-sigmoid functions are the most commonly used ones. Figure 4.3 illustrates the relationship between input and output for each transfer function respectively, where *y* represents the output from a neuron that receives a summed input of magnitude *z*. More specifically, these functions are defined by the equation 4.3, equation 4.4, equation 4.5 and equation 4.6 respectively.

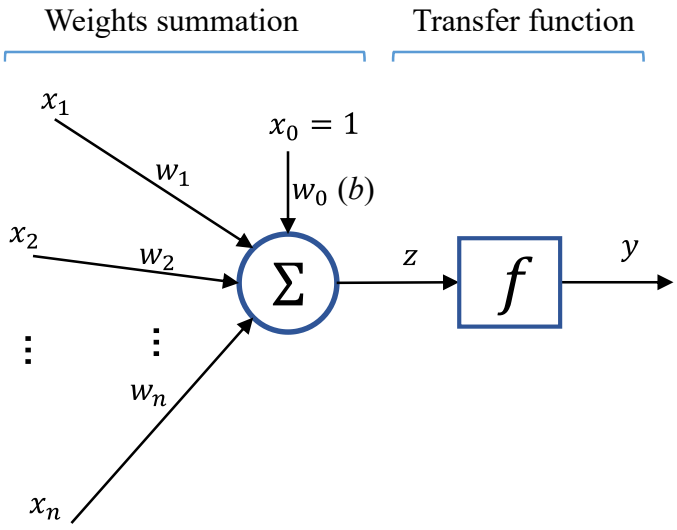

Figure 4.2 - A basic artificial neuron. Each neuron has several inputs and one output. It consists of weights summation (determines how the network inputs are combined inside a neuron) and activation function (determines the output).

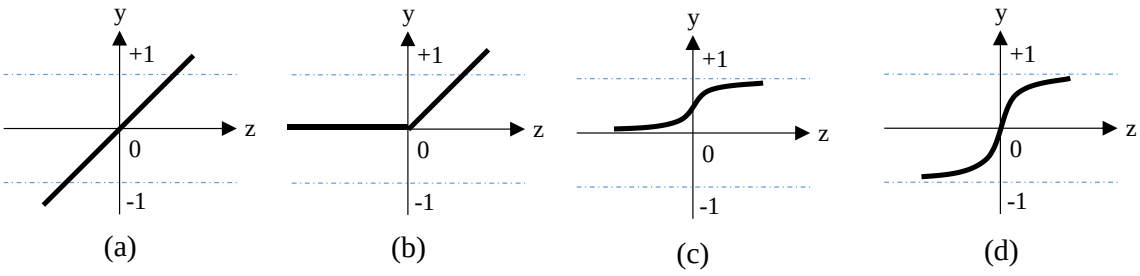

Figure 4.3 - Neuron transfer functions: (a) linear, (b) ReLU, (c) log-sigmoid, (d) tan-sigmoid.

$$y = z \tag{4.3}$$

$$y = \begin{cases} 0 \text{ for } z < 0 \\ z \text{ for } z \geq 0 \end{cases} \tag{4.4}$$

$$y = \frac{1}{1 + e^{-z}} \tag{4.5}$$

$$y = \frac{2}{1 + e^{-2z}} - 1 \tag{4.6}$$

4.2.3 Feedforward neural network

There are various types of ANN with different numbers of layers or complicated multi–input many directional feedback loops and layers, e.g., feedforward neural network, radial basis function network, recurrent neural network. Feedforward neural networks consist of a series of layers (i.e., input layer, hidden layer(s), and output layer), where each subsequent layer has a connection from the previous layer but do not form a cycle. In these networks, the information moves in only a forward direction, from the input layer, through the hidden layer(s) and to the output layer.

A multilayer perceptron (MLP), which is one of the most commonly used neural network, is a feedforward ANN model with one or more layers between inputs and outputs. Within a MLP, neurons of each layer are fully connected to the ones in the next layer. MLP has been widely used for function approximation, classification, recognition, and prediction.

Figure 4.4 demonstrates a typical structure of a MLP, which contains one input layer, one hidden layer, and one output layer. The input layer merely serves to pass the input vector to the network, so it plays no computational role. There can be multiple hidden layers, but only one input layer and one output layer.

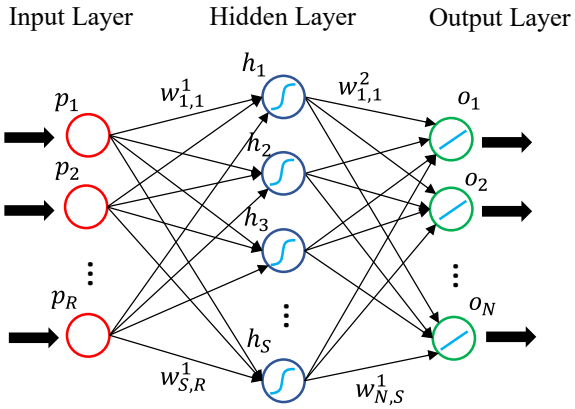

Figure 4.4 - A multilayer perceptron (MLP) structure, which contains one input layer, one hidden layer, and one output layer.

As shown in the figure 4.4, there are R input neurons, S neurons in the hidden layer, and N output neurons. The outputs of the hidden layer are the inputs of output layer. The final outputs of the MLP can be calculated by

$$O = f^2(W^2 f^1(W^1 P + B^1) + B^2) \tag{4.7}$$

where P=[p_1, p_2,…, p_R] is the input vector, O=[o_1, o_2,…, o_N] is the output vector, B^1=[b_1, b_2,…, b_S] and B^2=[b_1, b_2,…, b_N] are the bias vectors, W^1=[$w^1{}_{1,1}$, $w^1{}_{1,2}$,…, $w^1{}_{S,R}$] and W^2=[$w^2{}_{1,1}$, $w^2{}_{1,2}$,…, $w^2{}_{N,S}$] are the weight matrices, and f^1 and f^2 are the transfer functions of hidden layer and output layer respectively. Also, in this case, each neuron in the hidden layer uses a tan-sigmoid transfer function, and a linear function is used as the transfer function of the output layer.

By selecting suitable weights, biases, transfer functions, and the network architecture, a MLP is able to approximate any smooth, measureable function [103].

4.2.4 ANN training

The ANN architecture has to be determined for a particular application. Before the ANN can be used for calculating the outputs (e.g., FLIM analysis for this study), it has to be trained with (or learn from) a training dataset. The dataset comprises a sub-set of the data that is acquired from the target system or simulation.

The main objective in the ANN development is to find a function (which is determined by an optimal set of weight parameters w), such that f: $X \rightarrow Y$ closely matches the training dataset (approximates the original problem behaviour). There are three major categories of learning methods: supervised learning (each training sample is a pair consisting of an input vector and a desired output vector) [104, 105], unsupervised learning (it infers a function to describe hidden structure from "unlabeled" data) [92, 105], and reinforcement learning (it explores how software agents ought to take actions in an environment so as to maximize rewards) [104, 106]. In this case, the supervised learning is used to train the ANN, and the training dataset are example pairs, i.e., input features of the system x (histogram data), target data y (FLIM parameters) which has been predetermined by another method (e.g., simulation measurement).

Figure 4.5 demonstrates the diagram of supervised training. The learning algorithm attempts to improve the mapping implied by data (minimize the error between the ANN output and the target) through updating the weights of the network. Considering a simple ANN which has only two unknown parameters (e.g., two weights w_1 and w_2), the relationship between weights parameters and error (between the ANN output and training target) is given in figure 4.6. More specifically, this error plot is generated by varying the weights through all possible values. The error surface demonstrated in figure 4.6 has three local minima. The objective of training process is to find the best weights related to the global minimum, so that the ANN is able to imitate the target system as much as possible. However, in practice, it is not possible to generate such an error plot or find the best weights by observing, since the number of weights is very large in most cases. It is necessary to design an algorithm that can find the global or local minima.

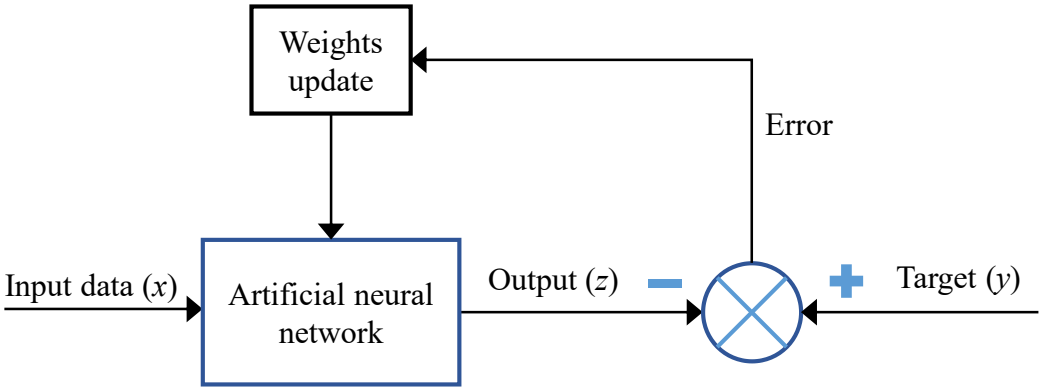

Figure 4.5 - A diagram of supervised learning. The learning algorithm attempts to improve the mapping implied by data through updating the weights of the network.

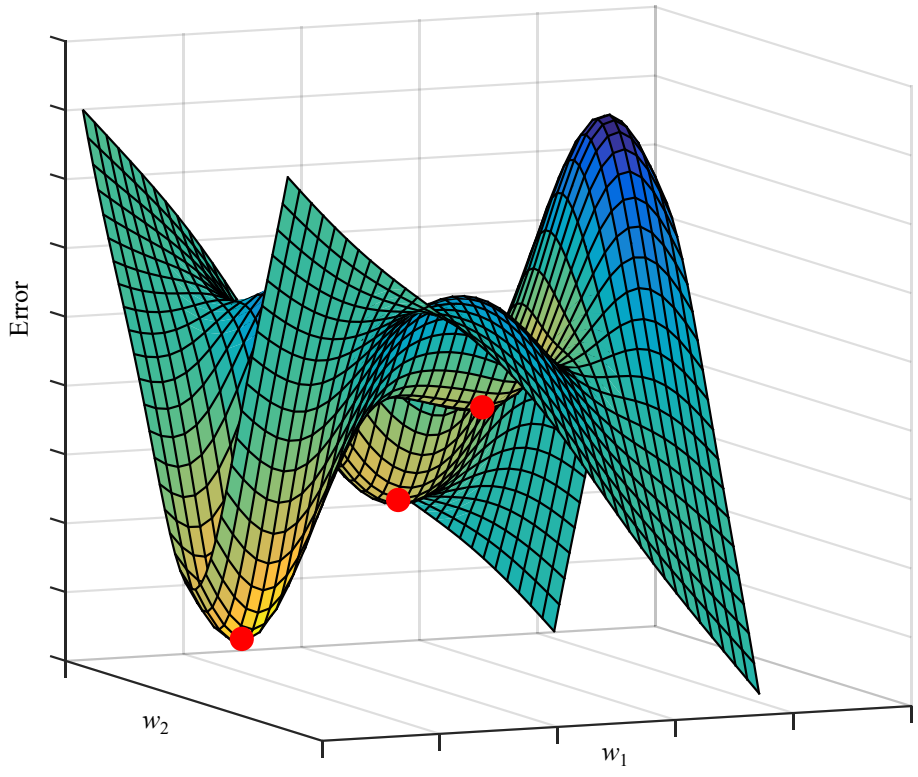

Figure 4.6 - The error surface (the relationship between weights parameters and error). Especially, the red points are the local minima. The objective of training process is to find the best weights related to the global minima, so that the ANN is able to imitate the target system as much as possible. However, in practice, it is not possible to generate such an error plot or find the best weights by observing, since the number of weights is very large in most cases. It is necessary to design an algorithm that can find the global or local minima.

During the training process, the performance of an ANN is usually evaluated by calculating the mean square error between neural network outputs and corresponding targets, which is given by

$$E = \frac{1}{2}\sum_{i=1}^{M}\sum_{j=1}^{N}(o_{i,j} - \beta_{i,j})^2 \tag{4.8}$$

where $\beta_{i,j}$ is the jth target of ith sample, $o_{i,j}$ is the jth neural network output for ith sample input as described in E.q. (4.7), M is number of training samples, and N is the number of neural network outputs.

Among the supervised training algorithms, backpropagation (BP) [107] is commonly used, and is used in conjunction with an optimization method such as gradient descent (GD-BP). GD-BP is a computationally straightforward algorithm for network training, and has been shown to perform adequately in many applications. It calculates the gradient $\frac{\partial E}{\partial \boldsymbol{W}}$ of the error function E, then the gradient is used by the optimization algorithms to update the weights. GD-BP follows the following steps:

1. Initialise the ANN weights,
2. Forward propagation of a training input vector through the neural network in order to generate the network output,
3. Calculate the error function E,
4. Backward propagation of error signal through the neural network in order to generate the partial derivative of the error with respect to a weight $w_{i,j}$ (the gradient of the weight),
5. To update the weight $w_{i,j}$, the old weight is subtracted by a fraction (learning rate) of the gradient,
6. Repeat steps 2–5 with next input vector of training sample, until the training process meets the convergence criteria.

This implementation of GD-BP learning procedure is known as online training where the weights are updated after each single sample (or pattern) has been presented. There is another training approach named batch training, where the summed error for all (or a set

of) patterns is used to update the weights. Although these two approaches are similar they can produce very different results, and they are optimized for different scenarios.

If the trained ANN never learns, it could be because the input data and target data are not strongly correlated (the input data does not contain the specific information required for deriving the desired output). Sometimes the network training does not converge, because of lack of sufficient training data. If the ANN still cannot provide satisfactory results, it is worth reviewing the architecture of the network, the transfer function, and the training algorithm.

4.2.5 Applications of ANNs

There are endless real world applications of ANNs. Some well-known ones including but are not limited to:

1) Function approximation and prediction

Feedforward ANNs are able to approximate highly non-linear functions between inputs and targets without prior knowledge of the relationship, which is required for traditional regression analysis methods. For example, ANNs have been successfully trained to model the relationship between hourly groundwater levels and various local variables, which are rainfall and evapotranspiration [108].

2) Image (data) compression

ANNs have been used for compression and decompression of data. For example, a compression technique for still digital images has been proposed with deep neural networks [109].

3) Financial prediction

The accurate prediction of stock price and bankruptcy in different industries is of a great concern to investors and creditors, banks, credit card companies and lending institutions. Neural networks are now trained to predict these financial features by selecting appropriate independent variables [110].

4) Signal processing

Neural networks have been employed to reduce noise. In 2013, an adaptive filtering approach based on discrete wavelet transform and artificial neural network was proposed for electrocardiogram (ECG) signal noise reduction [111].

5) Language processing and speech recognition

Applications, such as automatic language translation, automatic transcription, text-to-speech conversion, secure voice keyed locks, aids for the deaf, and natural language processing, have taken more and more advantage of ANN technologies. Geoffrey et al. proposed deep neural networks (DNNs) that can outperform Gaussian mixture models on a variety of speech recognition benchmarks [112].

6) Hand-written character recognition

Neural network based hand-written character recognition systems are now able to achieve high recognition accuracy. For example, a convolutional neural network based recognition framework can now deliver accuracy rate higher than 98% [113].

4.3 ANN-FLIM for tail fitting

Due to increasing demands for high-resolution, high-speed FLIM images, it is desirable to develop a high-speed FLIM data acquisition system and a fast FLIM analysis tool. Recent development of CMOS technologies have dramatically improved the measurements of photon events. On the other hand, these advanced systems, which generate massive data throughput making FLIM analysis even more challenging. In order to make a quick analysis, one simple solution is to ignore the effect of IRF by fitting the observed TCSPC histogram only in the tail region of the decay (it is known as “tail-fitting”) [114].

As discussed above, many research areas have been benefited from machine learning (ML) techniques, and recent advances in high performance computing with parallel processors have pushed them to another level. As an ML approach, ANN does not require

users to fully understand the complicated mathematical implications of histogram data. It can simply learn from the data by itself, and find out the links between inputs and outputs. Based on this outstanding feature, we apply it to simplify the FLIM analysis.

Figure 4.7 demonstrates the principle of the ANN-FLIM for tail-fitting FLIM analysis, where the input is the pre-processed FLIM histogram data and the output are the expected unknown parameters. FLIM images can be generated based on these parameters. After the ANN is well trained (with synthesized data sets), it can obtain target parameters from the input histograms. The lifetime estimations do not need any convolution process. ANN-FLIM approaches simply preclude all complicated computations from image generation, and therefore they can achieve high-speed easily.

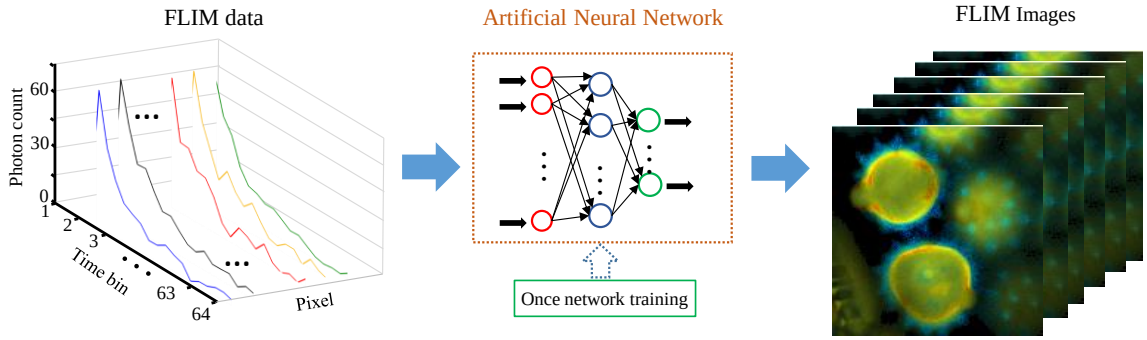

Figure 4.7 - Principle of the ANN-FLIM analysis: Selected photon counts of each pixel are used as the rate inputs to an ANN that approximates the unknown function from the high-dimensional space of photon count histograms to decay constants and other parameters of the lifetime function, from which FLIM images can be generated.

4.3.1 FLIM Model

In TD FLIM, the measured fluorescence decay histogram $y(t)$ is equivalent to the sample's intrinsic fluorescence decay convoluted with the excitation profile (i.e., IRF) [5]. However, as for tail-fitting approaches, we assume the measured densities are equivalent to intrinsic decay. In this chapter, we assume the decay is bi-exponential:

$$y(t) = Kf_D \exp(-t/\tau_F) + K(1-f_D)\exp(-t/\tau_D) \tag{4.9}$$

where K is pre-scalar, f_D is the proportion, h is the bin width, and τ_F, τ_D are the lifetimes.

For a TCSPC based FLIM system, the decay signal Eq. (4.9) can be re-written as a discretized model:

$$y(k) = Kf_D \exp[-(k-1)h/\tau_F] + K(1-f_D)\exp[-(k-1)h/\tau_D] \quad (4.10)$$

4.3.2 ANN architecture

ANNs mimic how human neurons respond to new external stimuli based on previous experiences [101, 115]. In order to implement a high speed algorithm, which does not require iterative calculation procedures, a simple feed-forward ANN structure is introduced for this study. As shown in figure 4.8, it contains one input layer, one output layer and two hidden layers [116]. The neurons of the output layer have linear transfer functions, whereas the other neurons in the input and hidden layers are configured with sigmoid transfer functions. As one of the most common architectures, the feed-forward network only allows the neurons in each layer to transfer information to neurons in the next layer (from the input layer to the hidden layers, and then ultimately to the output layer). The number of neurons in the input layer depends on the number of time bins in the histogram (64 time bins in this study). As for the output layer, four neurons are used (more neurons can be used in the future) to generate K, f_D, τ_F, and τ_D, respectively. The two hidden layers implement the transformation from 64 inputs to the four outputs, which includes implicitly the underlying relationship among the four outputs. To be more specific, each layer has 64, 32, 128, 4 neurons, respectively (the discussion about how to choose the architecture of ANN will be further demonstrated in section 6.4.3).

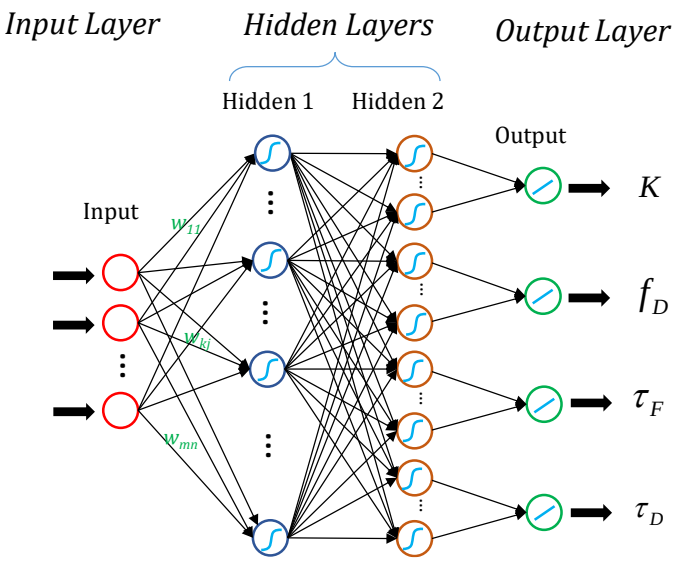

Figure 4.8 - The architecture of the ANN (Modified from [116]). The ANN contains 2 hidden layers (with sigmoid transfer function based neurons), one input layer and one linear output layer. To be more specific, each layer has 64, 32, 128, 4 neurons, respectively.

4.3.3 ANN Training

Before the ANN can be used for FLIM analysis, it has to be trained with appropriate training samples, i.e., fluorescence decay histograms (input) and corresponding parameters (output). This procedure contains three steps:

1) Sample Preparation: The weights w_{kj} encode the mapping of histograms to outputs and are initially unknown. They can be deduced from a large number of synthesized sample sets, i.e. synthesized histograms and matching vectors $\alpha = [K, f_D, \tau_F, \tau_D]$. Ideally, one would like to train the network with original α (targets of training samples) and noisy histograms (inputs of training samples) generated based on this α, and we originally have tried to do this. However, the fully trained ANNs failed to recover the true α in the majority of cases. This is compatible with the observation that other leading FLIM algorithms are in many circumstances also unable to recover the true α due to ambiguities caused by noise. As the next best goal to recovering the true parameters, we now train the ANN to recover the results of a maximum likelihood estimation (MLE) of the parameters obtained from the same histogram. As demonstrated in Table 4.1, it is much more difficult to train the network with training samples that use original α as targets (it requires 3-fold

more time to train the network). One can also learn that the MLE based ANN training has significantly improved the overall performance of network. We observed that the output error of MLE based network is 2.9%, while the error of the original α based network was 5.5%.

Table 4.1 Comparison of ANNs trained with different samples

Training targets	**Number of samples**	**Hardware**	**Training time (h)**	**Relative performance (error percentage)**
Original α	120,000	Intel(R) Xeon(R) E5-2609 v2 processor with 32 GB memory	13.9	5.5%
MLE based			4.1	2.9%

Figure 4.9 demonstrates how to prepare sufficient training samples for a tail-fitting FLIM system based on synthesized data [116]. First, a set of α vectors from within the working range are chosen (the range of each parameter is usually known in advance) to generate FLIM decays for training. It is important to cover the relevant areas of the parameter space well. This improves the generalization of the ANN, so that the ANN can work equally well across the whole working range of the parameters. To do so, we use the Hammersley quasi-random sequence, which is a low-discrepancy sampling (LDS) method and can efficiently generate a low discrepancy sequence [117, 118]. Next, the theoretical fluorescence histogram $y(k)$ is generated based on α by following the decay model as described in Eq. (4.9). Then the TCSPC decay signals can be simulated by combining Poisson noise with the theoretical histogram values, resulting in the final histogram as illustrated by the red curve in the middle of figure 4.9. The noisy TCSPC decays are fed to the maximum likelihood estimation (MLE) [37, 119, 120], which is known as one of the best FLIM analysis algorithms. If α is the initial value for MLE, then the results ($\alpha^* = [K^*, f^*_D, \tau^*_F, \tau^*_D]$) of the MLE are used as the training targets of the ANN. In brief, to perform MLE, the definite integral of fluorescence decay $\Lambda(t)$ is given by Eq. (4.11), and the expected value EN_i of each time bin can be obtained as in Eqs (4.12). Therefore, the likelihood of the observed histogram decay is given by Eq. (4.13), and the

preferable targets of the training samples can be acquired based on maximizing this likelihood function.

$$\begin{aligned}\Lambda(t) &= \int_0^t f(t)dt \\ &= -A(\tau_F f_D \exp(-t/\tau_F) + \tau_D(1-f_D)\exp(-t/\tau_D))\Big|_0^t,\end{aligned} \tag{4.11}$$

$$\begin{aligned}EN_i &= \Lambda(ih) - \Lambda((i-1)h) \\ &= K[\tau_F f_D \exp(-ih/\tau_F)(\exp(h/\tau_F)-1) \\ &\quad + \tau_D(1-f_D)\exp(-ih/\tau_D)(\exp(h/\tau_D)-1)],\end{aligned} \tag{4.12}$$

$$L = \prod_{i=1}^{m} \frac{EN_i^{N_i} \exp(-EN_i)}{N_i!}, \tag{4.13}$$

where N_i is the photon count of the ith TCSPC time bin, m is the number of time bins and h is the bin width.

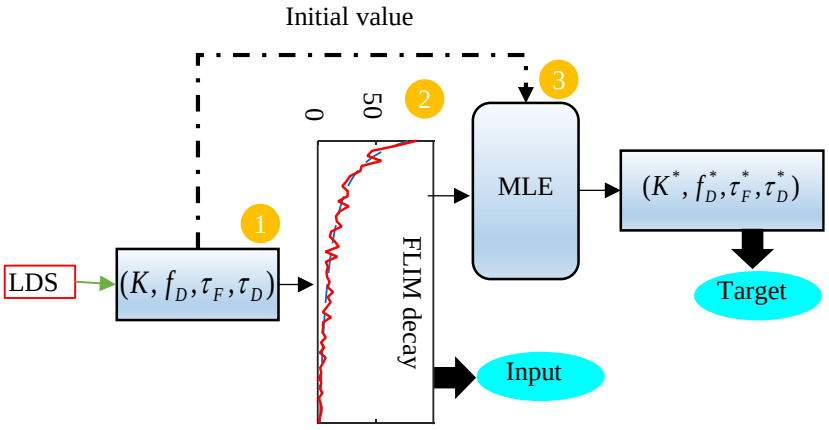

Figure 4.9 - Preparation of the training samples (Reproduced from [116]). First, a set of α vectors from within the working range are chosen. Next, the theoretical fluorescence histogram is generated based on α by following the decay model. Then the TCSPC decay signals can be simulated by combining Poisson noise with the theoretical histogram values. The noisy TCSPC decays are fed to the maximum likelihood estimation, and the results of the MLE are used as the training targets of the ANN.

2) Sample Pre-processing: Since the total photon counts can vary significantly across pixels, for a certain time bin the difference between highest and lowest counts can be huge. The huge difference increases the difficulty of training a well-performing ANN.

The majority of the counts may be only distributed in a narrow range, but the training process tries to equalize all the samples. Besides, since the sigmoid transfer function is used in the hidden layers, when the input is large (i.e., greater than 3) this function becomes essentially saturated [101, 121]. The gradients will be very small which slows the network training process [101, 121]. It is therefore necessary to rescale the decays, and make the count distributions as even as possible. Here, we normalized the new inputs of the ANN (i.e., each photon count was divided by the total count of the corresponding pixel.). Moreover, before the training samples are used to train the network, data-scaling was performed for the targets as well. As described in the next sub-section, the network training process minimizes the mean square error between the targets and network outputs. The errors from the target parameters with smaller magnitudes have less influence or can be ignored, and these parameters generated by the ANN have very low accuracy. To overcome this problem, a MATLAB function called "mapminmax" can be used, which maps the values to [-1, 1] [101].

3) Network Training: The goal of ANN training is to find better weights w_{ij} (similar to the way how neurons connect and interact), which are initially unknown. This process tries to minimize the error between the outputs of the network and the target parameters α^* by iteratively updating the weights. As demonstrated in Eq. (4.14), the normalized mean squared error is used as the error function. Weight updates can be accomplished with the supervised backpropagation (BP) learning method [122]. BP methods in conjunction with gradient descent optimization algorithm [101] were applied in the training process. More specifically, from the Matlab neural network toolbox (other machine learning framework such as Tensorflow has also been tested, but for this work Matlab provides better results), the scaled conjugate gradient BP function ("trainscg") is used to train the network, as it is much faster than the Levenberg-Marquardt BP function when trained with larger sample sets. Moreover, the trainscg function also supports graphics processing unit (GPU) acceleration, which gives a speed-up factor of 10.

$$F_{mse} = \frac{1}{N}\sum_{1}^{N}[w_i(\alpha_{Ti} - \alpha_{Oi})^2] \tag{4.14}$$

4.3.4 Lifetime Calculation

Once being trained, the ANN is able to calculate the FLIM parameters in a straightforward manner, without any iterations or initial conditions. For example, for a ANN with 2 hidden layers, the output can be described as $a^3 = \mathbf{f}^3(\mathbf{W}^3\mathbf{f}^2(\mathbf{W}^2\mathbf{f}^1(\mathbf{W}^1\mathbf{p}+\mathbf{b}^1)+\mathbf{b}^2)+\mathbf{b}^3)$, where $\mathbf{p}$ is the input vector, $\mathbf{b}^1$, $\mathbf{b}^2$, and $\mathbf{b}^3$ are the bias vectors, $\mathbf{W}^1$, $\mathbf{W}^2$, and $\mathbf{W}^3$ are the weight matrix, and $\mathbf{f}^1$, $\mathbf{f}^2$, and $\mathbf{f}^3$ are the transfer functions vectors of the first hidden layer, second hidden layer, and output layer, respectively. As demonstrated in figure 4.10, the ANN-FLIM mapping contains tail selecting (finding the tail region of the original decay), pre-processing, neural network computation, and post-processing. As mentioned in Section Sample Pre-Processing, the histogram is scaled before being fed to the input neurons. Once the ANN is trained, it is deployed in the FLIM system, with the same architecture and the preferred weights, and the outputs can be calculated accordingly. Since the ANN is trained to calculate the rescaled FLIM parameters, the final results can be acquired by applying the same mapminmax-scaler used in the ANN training to the outputs of the ANN.

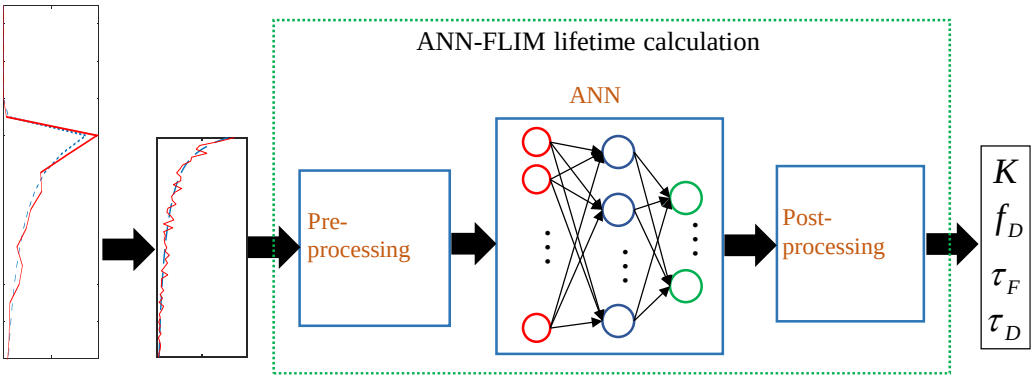

Figure 4.10 - ANN-FLIM lifetime calculation procedure. The ANN-FLIM mapping contains tail selecting, pre-processing, neural network computation, and post-processing.

4.4 ANN-FLIM including IRF

Most traditional algorithms require time-consuming iterative computations, especially for analysing multi-exponential FLIM data with IRF. Fortunately, ANNs are able to learn

from the samples and build the connections between inputs and outputs even when they are very complicated. This leads us to design a new algorithm that can deal with IRF for FLIM analysis based on ANNs.

Figure 4.11 demonstrates the principle of the ANN-FLIM, where the input is the FLIM histogram data and the output are the expected FLIM parameters. Finally, FLIM images are generated by FLIM image generator (which defines the corresponding colour of FLIM parameters) based on these parameters. Different from the tail-fitting ANN, this new implementation allows us to consider the whole histogram. After the ANN is well trained, it can obtain target parameters from the input histograms. The lifetime estimations do not need any deconvolution or reconvolution process. ANN-FLIM approaches simply preclude all complicated computations from image generation (the ANN includes the entire mapping from the IRF-dependent histogram to the underlying FLIM lifetimes), and therefore they can achieve high-speed easily.

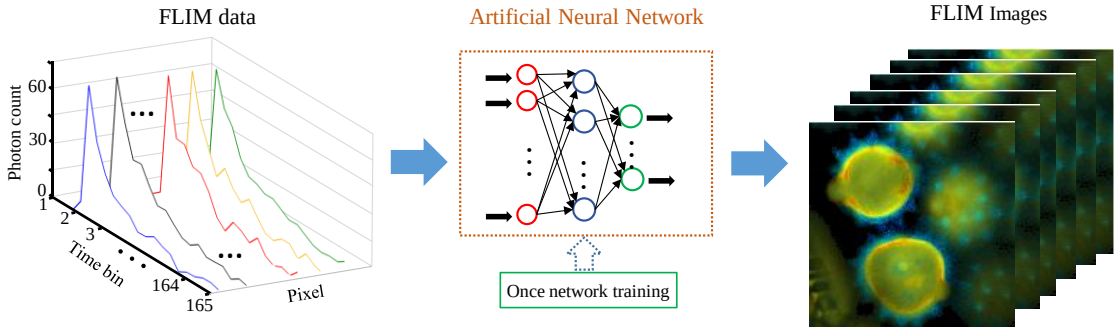

Figure 4.11 - Principle of the ANN-FLIM analysis: Photon counts of each pixel are used as the rate inputs to an ANN that approximates the unknown function from the high-dimensional space of photon count histograms to decay constants and other parameters of the lifetime function, before FLIM images can be generated.

4.4.1 FLIM Model

In TD FLIM, the measured fluorescence decay histogram $y(t)$ is equivalent to the sample's intrinsic fluorescence decay convoluted with the excitation profile (i.e., IRF) $E(t)$ [5]:

$$y(t) = E(t) \otimes I(t) = \int_0^t E(t-z) \cdot I(z) dz \tag{4.15}$$

where I(t) is the intrinsic fluorescence density function.

For a TCSPC based FLIM system, the decay signal Eq. (4.15) can be re-written as a discretized model:

$$y(k) = \sum_{i=1}^{k} E(k-i) \cdot I(i) + \varepsilon(k) \tag{4.16}$$

where $\varepsilon(k)$ is the Poisson noise introduced during the measurement.

In this chapter, we assume the decay is bi-exponential:

$$I(k) = Kf_D \exp[-(k-1)h/\tau_F] + K(1-f_D)\exp[-(k-1)h/\tau_D] \tag{4.17}$$

where K is pre-scalar, f_D is the proportion of donor in interaction, h is the bin width, and τ_F, τ_D are the lifetimes.

4.4.2 ANN architecture

The same as the tail-fitting ANN, to achieve high-speed analysis, a feedforward ANN is also used. In such an ANN, connections between neurons do not form a cycle [101]. However, to learn the more complex function including the IRF as described in the previous section, a more powerful ANN is introduced. As demonstrated in figure 4.12, the ANN contains 4 hidden layers (with sigmoid transfer function based neurons), one input layer and one linear output layer. To be more specific, each layer has 128, 32, 32, 64, 64, 4 neurons, respectively (the discussion about how to choose the architecture of ANN will be further demonstrated in section 6.4.3.3).

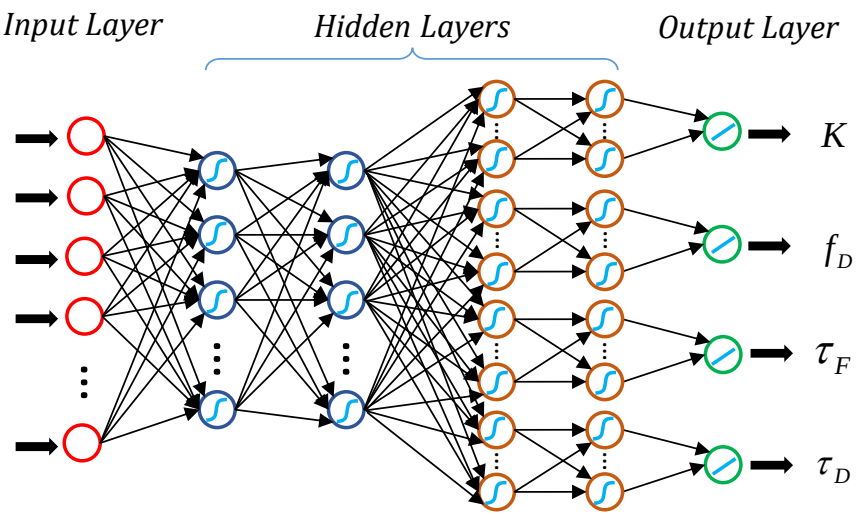

Figure 4.12 - The architecture of the ANN. The ANN contains 4 hidden layers (with sigmoid transfer function based neurons), one input layer and one linear output layer. To be more specific, each layer has 128, 32, 32, 64, 64, 4 neurons, respectively.

4.4.3 ANN Training

The same as previously described, the ANN has to be trained with appropriate training samples, i.e., fluorescence decay histograms (input) and corresponding parameters (output). Similar to the tail-fitting ANN, this procedure also contains three steps:

1) Sample Preparation: Figure 4.13 demonstrates how to prepare sufficient training samples for a FLIM system based on synthesized data. First, the IRF of the target FLIM equipment is acquired [19]. Then, as reported previously [116] and above, a set of vectors $\alpha = [K, f_D, \tau_F, \tau_D]$ is chosen to generate FLIM decays for training, with the underlying assumption that the range of each parameter is already known. It is important to improve the generalization of the ANN, so that the ANN can work equally well across the whole working range of the parameters. To do so, we use the Hammersley quasi-random sequence, which is a low-discrepancy sampling (LDS) method and can efficiently generate a low discrepancy sequence [117, 118]. Next, the intrinsic decay function $I(t)$ is generated based on α. In addition to what was done previously, we then convolve $I(t)$ with the IRF in order to obtain the theoretical fluorescence histogram, $y(t)$. Then the TCSPC decay signals can be simulated by combining Poisson noise with the theoretical histogram values, resulting in the final histogram as illustrated by the red curve in the

middle of figure 4.13. The noisy TCSPC decays are fed to the maximum likelihood estimation (MLE) block, with α as the initial value, and the results ($\alpha^* = [K^*, f^*_D, \tau^*_F, \tau^*_D]$) of the MLE are used as the training targets of the ANN as in previous section but now on the more realistic histogram including the IRF. Finally, the photon counts in the first few bins before the rising part of the histogram are very close to zero. Since these counts make very little contribution to lifetime calculation and mainly contain noise, we can neglect them in order to increase processing speed. We found that the first 38 bins can be disregarded in our case (our approach does not need to perform this step, but to simplify the estimations we neglect them here). The photon counts of the remaining time bins are the inputs to the ANN in the training samples.

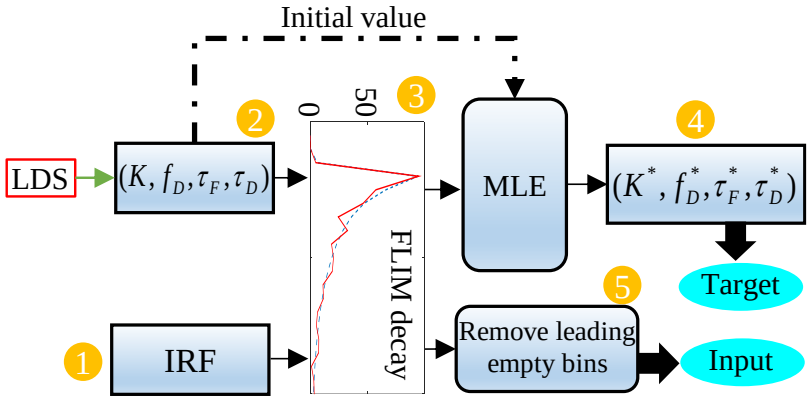

Figure 4.13 - Preparation of the training samples. First, the IRF of the target FLIM equipment is acquired. Then, a set of vectors $\alpha = [K, f_D, \tau_F, \tau_D]$ is chosen to generate FLIM decays for training. Next, the intrinsic decay function $I(t)$ is generated based on α. Then the TCSPC decay signals can be simulated by combining Poisson noise with the theoretical histogram values, resulting in the final histogram as illustrated by the red curve in the middle of figure. The noisy TCSPC decays are fed to the maximum likelihood estimation (MLE) block, with α as the initial value, and the results ($\alpha^* = [K^*, f^*_D, \tau^*_F, \tau^*_D]$) of the MLE are used as the training targets of the ANN. Finally, the photon counts in the first few bins before the rising part of the histogram are neglected them in order to increase processing speed.

2) Sample Pre-processing: The same as described in the tail-fitting section, it is necessary to rescale the decays, and make the count distributions as even as possible. Instead, here, we applied logarithmic scaling (base 10) to the new inputs of the ANN. To avoid invalid operations, each count was incremented by 1. Moreover, before the training samples are used to train the network, data-scaling was performed for the targets as well. As for the target parameters, a MATLAB function called “mapminmax” is used to map the values to [-1, 1].

3) Network Training: The ANN training is to find better weights (similar to the way how neurons connect and interact) that can minimize the error between the outputs of the network and the target parameters α^*. As I have described previously, the mean square error function is used as the performance function. BP methods in conjunction with gradient descent optimization algorithm [101] were applied in the training process. Also, from the Matlab neural network toolbox, the scaled conjugate gradient BP function ("trainscg") is used to train the network, as it is much faster than the Levenberg-Marquardt BP function when trained with larger sample sets. The same as previously described, the GPU accelerated trainscg function gives a speed-up factor of 10 more than the CPU-only training procedure.

4.4.4 Lifetime Calculation

After being trained, the ANN is able to calculate the FLIM parameters in a straightforward manner, without any iterations or initial conditions. As demonstrated in figure 4.14, the ANN-FLIM mapping contains pre-processing, neural network computation, and post-processing. As mentioned in Section Sample Pre-Processing, the histogram is scaled before being fed to the input neurons. Once the ANN is trained, it is deployed in the FLIM system, with the same architecture and the preferred weights, and the outputs can be calculated accordingly. Since the ANN is trained to calculate the rescaled FLIM parameters, the final results can be acquired by applying the same mapminmax-scaler used in the ANN training to the outputs of the ANN.

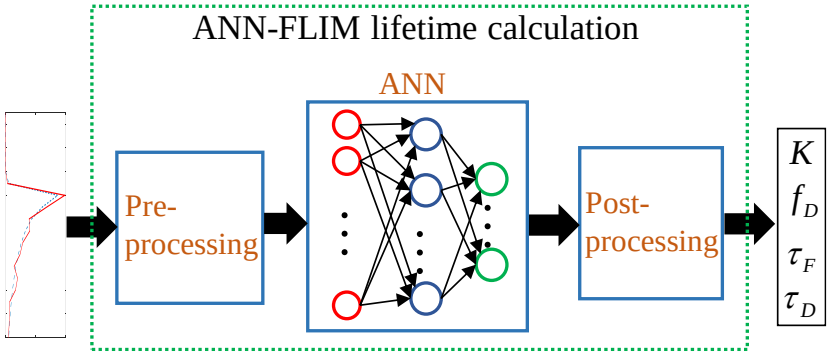

Figure 4.14 - ANN-FLIM lifetime calculation procedure. The ANN-FLIM mapping contains pre-processing, neural network computation, and post-processing.

4.5 Summary

This chapter demonstrates the configuration of the proposed artificial neural network base FLIM analysis method. More importantly, based on the features of ANNs, two feedforward ANNs have been introduced for tail-fitting FLIM analysis and IRF included FLIM analysis. Most of traditional methods require time-consuming iterative processes, which make building a high-speed FLIM analysis very challenging. The ANN-FLIM methods presented in this chapter, however, are simple and able to generate fast lifetime imaging without sacrificing performance with costly iterative estimators. This chapter also explain how to build and use ANN-FLIM methods for FLIM analysis.

Chapter 5 GPU accelerated ANN-FLIM analysis

5.1 Overview

In order to unveil dynamic cellular activities, it is critical to design and implement high-speed FLIM analysis system. So far, both state-of-art high-speed parallel processor and new ANN-FLIM algorithms have been demonstrated. The next step is to implement ANN-FLIM algorithms on GPUs so that they can harvest the power of parallel processors and maximize the performance.

In this chapter, first, the GPU implementations of ANN-FLIM algorithms are demonstrated. For the GPU acceleration of ANN-FLIM methods, the key is to optimise the matrix multiplications. Then the ANN-FLIM algorithms will be evaluated by comparing with traditional algorithms, such as LSM, LSIR. Moreover, to further verify the ability of GPU accelerated ANN-FLIM methods, experimental data evaluation process has also been included in this chapter.

5.2 GPU acceleration of ANN-FLIM methods

The essential operation in an ANN is the inner-product between an input vector and a weight vector in each layer [123, 124]. In order to maximize the parallelization of lifetime calculations and utilization of a GPU, input vectors (histogram data) of each pixel within the same FLIM image and weight vectors of the same layer are accumulated. This configuration allows us to transform inner-product operations into matrix multiplications, which are more appropriate for GPU implementation. The computation of an ANN layer

is then followed by a bias factor addition and sigmoid operation. Although the network structure of different ANNs for FLIM analysis varies as regards the number of neurons of each layer, and number of hidden layers, each layer performs the same matrix multiplication, followed by a non-linear function [123]. Figure 5.1 demonstrates the computation flow of neural networks on GPU.

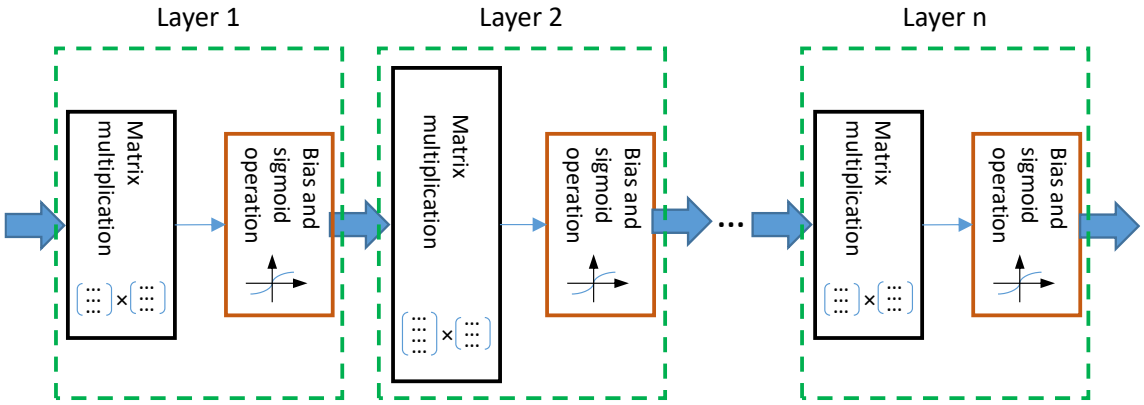

Figure 5.1 - The computation flow of neural networks on GPU. Each layer contains a matrix multiplication and a non-linear operation, and the results of the network is calculated layer by layer.

5.2.1 CUDA matrix multiplication

As demonstrated above, by accumulating the input and weight vectors of each layer, the inner-product operations can be replaced with a matrix multiplication. As such, inner-product operations can be re-described as follows:

$$P = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1N} \\ p_{21} & p_{22} & \cdots & p_{2N} \\ \cdots & \cdots & \cdots & \cdots \\ p_{M1} & p_{M2} & \cdots & p_{MN} \end{bmatrix}, W = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1L} \\ w_{21} & w_{22} & \cdots & w_{2L} \\ \cdots & \cdots & \cdots & \cdots \\ w_{N1} & w_{N2} & \cdots & w_{NL} \end{bmatrix}, \tag{5.1}$$

$$J = P \times W, \tag{5.2}$$

where P, W, and J are the input, weight, and intermediate matrices respectively. M, N, L are the number of pixels, input numbers, and neuron number of the layer respectively. In addition, p_{ij} is the jth input of ith pixel, and w_{ij} is the weight of jth neuron for ith input.

Matrix multiplication is an essential building block for numerous numerical algorithms, and it has been implemented and optimized in CUDA by NVIDIA [68]. It is important to use shared memory to improve the global memory load efficiency in matrix multiplication, because reading data from global memory (FLIM histogram data is kept here) is very expensive.

Figure 5.2 demonstrates a CUDA implementation of matrix multiplication [68], where input and intermediate matrices are kept in global memory, and weight matrix is kept in constant memory. In this implementation, J is comprised of a number of sub-matrix J_{sub}, which is generated by each thread block, and each thread within the block is responsible for computing one element of J_{sub} (red square in J_{sub} as shown in figure 5.2). This implementation strategy was designed based on the features of the GPU device (e.g., maximum threads in a block, size of shared memory). As illustrated in the figure, J_{sub} is the output matrix of two rectangular matrices, i.e., the sub-matrix of P of dimension (P.width, BLOCK_SIZE) and the sub-matrix of W of dimension (BLOCK_SIZE, P.width). In order to optimize the CUDA program and fit into the device's resources, these two rectangular matrices are divided into as many square matrices of dimension BLOCK_SIZE as necessary. As a result, J_{sub} is acquired as the sum of the products of divided square matrices. Before the computation, the data of corresponding square matrices is loaded from global and constant memory to shared memory, and this procedure allows us to significantly reduce the traffic of memory transmission. Then each thread completes each inner-product operation, and by accumulating a certain number of corresponding inner-products the final results of J_{sub} can be acquired.

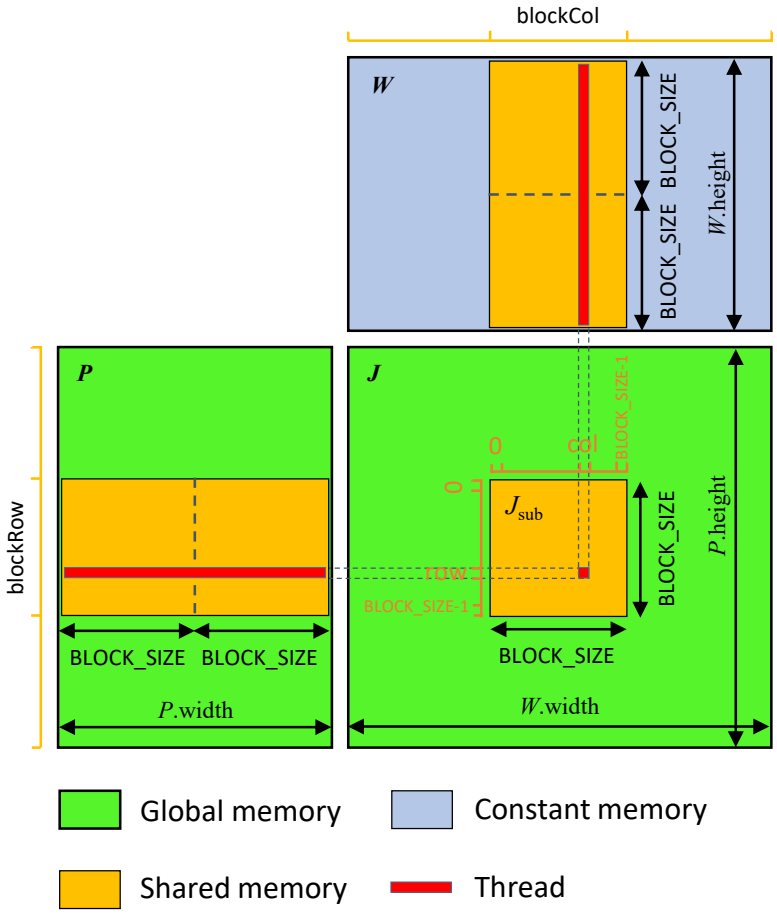

Figure 5.2 - A CUDA implementation of matrix multiplication with shared memory (Reproduced from [68]). J_{sub} is the output matrix of two rectangular sub matrices of P and W. In order to optimize the CUDA program and fit into the device's resources, these two rectangular matrices are divided into as many square matrices of dimension block_size as necessary. As a result, J_{sub} is acquired as the sum of the products of divided square matrices.

5.3 Evaluation of ANN-FLIM algorithms

To demonstrate the performance of the proposed ANN-FLIM algorithms, both of them will be evaluated by comparing with traditional algorithms based on synthesized data. The results are based on a NVIDIA TESLA K40 GPU and an OpenMP CPU implementation on an Intel(R) Xeon(R) E5-2609 v2 processor with 4 cores.

5.3.1 ANN-FLIM for tail-fitting

The proposed ANN-FLIM algorithm is compared to the widely used LSM method (nonlinear least square routine [125] with the Levenberg–Marquardt algorithm) using Monte-Carlo simulations. In the simulations, the number of time bins is m = 64, and the bin width is h = 333ps. In order to improve the performance of ANN-FLIM, it is better to specify the range of each FLIM parameter as shown in Table 5.1. All the training samples were generated based on the parameters within the corresponding ranges, and this configuration allows the ANN to provide optimized performance when actual data fall into these ranges. For this study, we chose 7000 α vectors using the LDS method, and generated 30 histograms by adding Poisson noise on theoretical histogram for each vector. In total, we created 120,000 training samples for ANN training. The training procedure took about 4 hours.

Table 5.1 Simulation setup of ANN-FLIM

	K	f_D	τ_F (ns)	τ_D (ns)
Range	[7.5, 200]	[0, 1]	[0.02, 1]	[1, 4]

In order to make the simulationmore realistic, each synthesized histogram has a photon count of less than 900 total photons (in the real applications, it is of significance to design an algorithm that requires less photons). Monte-Carlo simulation was done by comparing the results based on a group of selected α vectors, and we generated 1000 histograms by adding Poisson noise for each α vector. Figures 5.3(a)-(b), 5.4(a) -(b) and 5.5(a) -(b) show the precision plots of these two algorithms under the same configurations. In these results, F-value (a figure-of merit that can quantify the photon economy [126]) [35, 127] is used to describe the precision, where $F = N_C^{1/2}\,\sigma_g/g$, N_C is the photon counts of each histogram, σ is the standard deviation, and $g = f_D$, τ_F, or τ_D. Whereas, figures 5.3(c)-(d), 5.4(c)-(d) and 5.4(c)-(d) show the bias plots, for τ_F, f_D and τ_D, respectively. For figures 5.3 and 5.4 τ_D = 2.5ns, and for figures 5.5 τ_F = 0.58ns. For both precision and accuracy performance, the smaller result means the better performance. For example, as one can find from figure 5.3(a)-(b) that most of the area is lower than 14 for ANN-FLIM, while it is only around half of the area for LSM, and this means ANN-FLIM has higher precision. In addition,

the green areas illustrated in figures 5.3(e) and (f), 5.4(e) and (f), and 5.5(e) and (f) highlighted the areas that the ANN has better performance than LSM. From these figures, one can learn that the optimized regions (for the *F*-value and the bias) of LSM are different from those of ANN. Figures 5.3(a)-(b) and (e), 5.4(a)-(b) and (e), and 5.5(a) -(b) and (e) show that ANN can provide a wider optimized area for the *F*-value for all τ_F, τ_D, and f_D. Figures 5.3(c)-(d) and (f), 5.4(c)-(d) and (f), and 5.5(c)-(d) and (f) show that LSM produces slightly less biased τ_F, whereas ANN offers wider optimized areas for both τ_D, and f_D. Although there are differences between them, their estimations are of the same order.

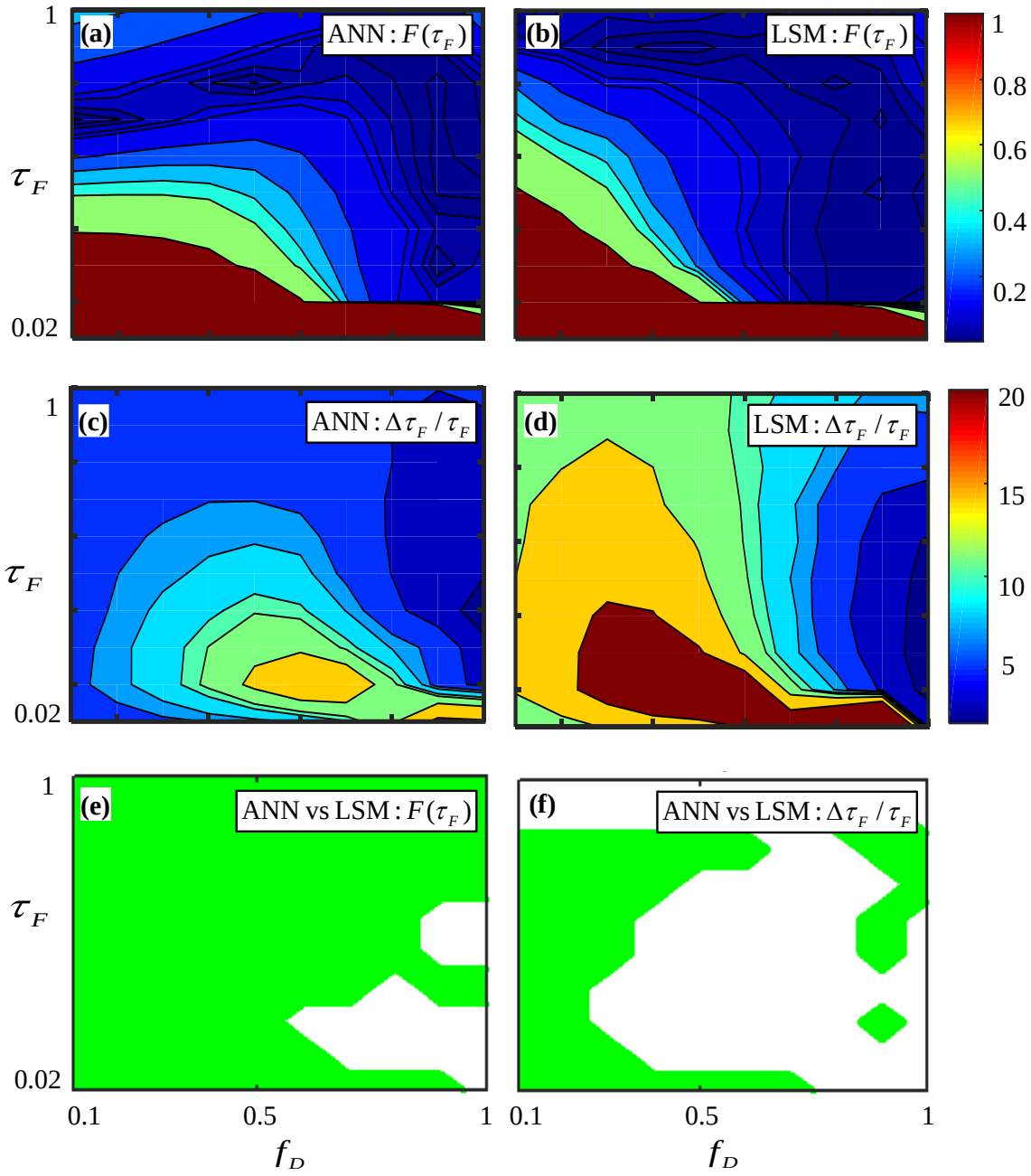

Figure 5.3 - Performances of τ_F calculated by ANN and LSM: (a) and (b) *F*-value; (c) and (d) the bias; (e) areas where ANN has better precision performance (green areas); (f) areas where ANN has better accuracy performance (green areas). For both precision and accuracy performance, the smaller result means the better performance. From these figures, one can learn that the optimized regions (for the *F*-value and the bias) of LSM are different from those of ANN. Figures 5.3 (a), (b) and (e) show that ANN can provide a wider optimized area for the *F*-value for all τ_F. Figures 5.3(c), (d) and (f) show that LSM produces slightly less biased τ_F. Although there are differences between them, their estimations are in the same order.

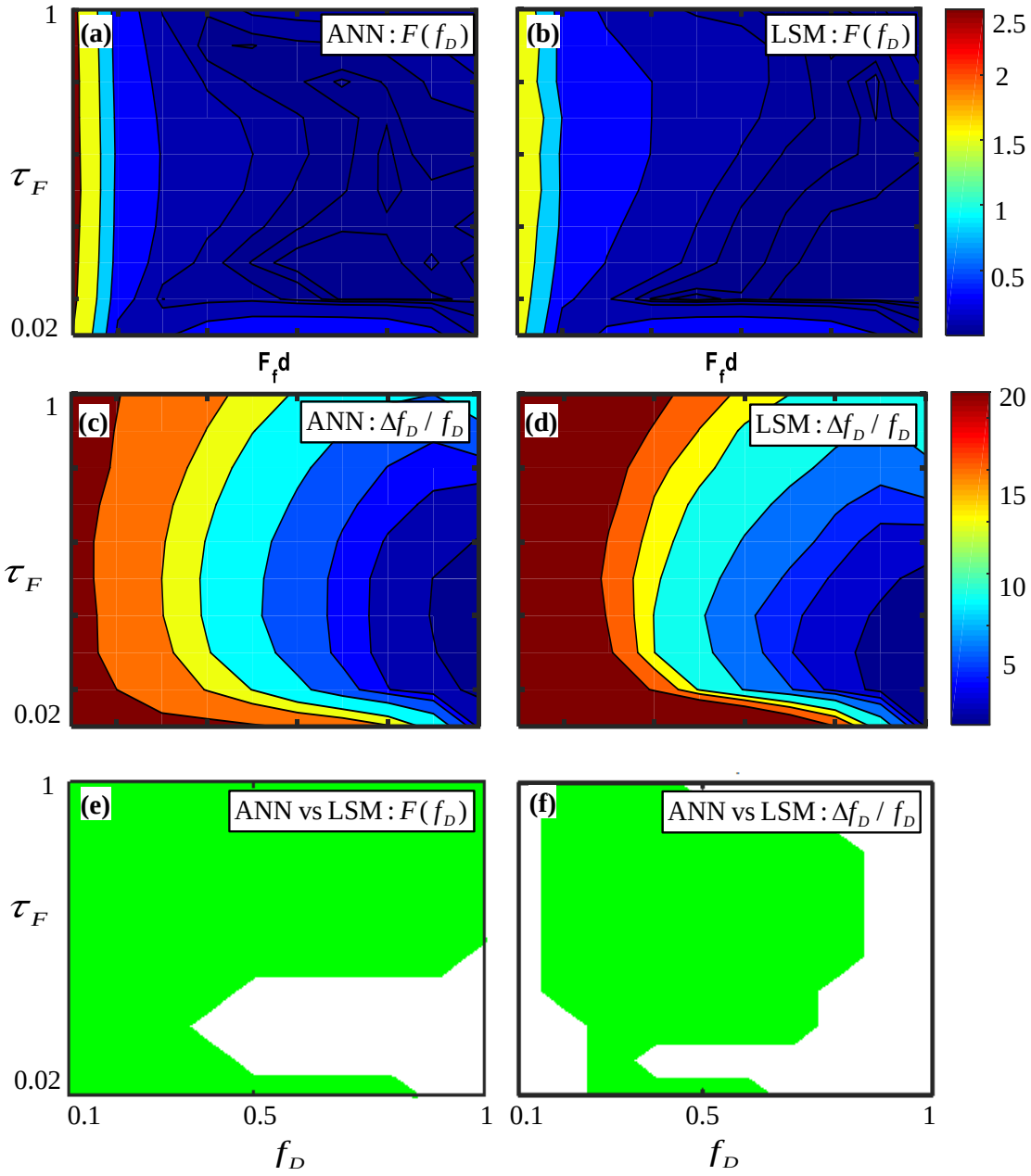

Figure 5.4 - Performances of f_D calculated by ANN and LSM: (a) and (b) F-value; (c) and (d) the bias; (e) areas where ANN has better precision performance (green areas); (f) areas where ANN has better accuracy performance (green areas). For both precision and accuracy performance, the smaller result means the better performance. From these figures, one can learn that the optimized regions (for the *F*-value and the bias) of LSM are different from those of ANN. Figures 5.4 (e) and (f) show that ANN can provide a wider optimized area for both *F*-value and bias for f_D.

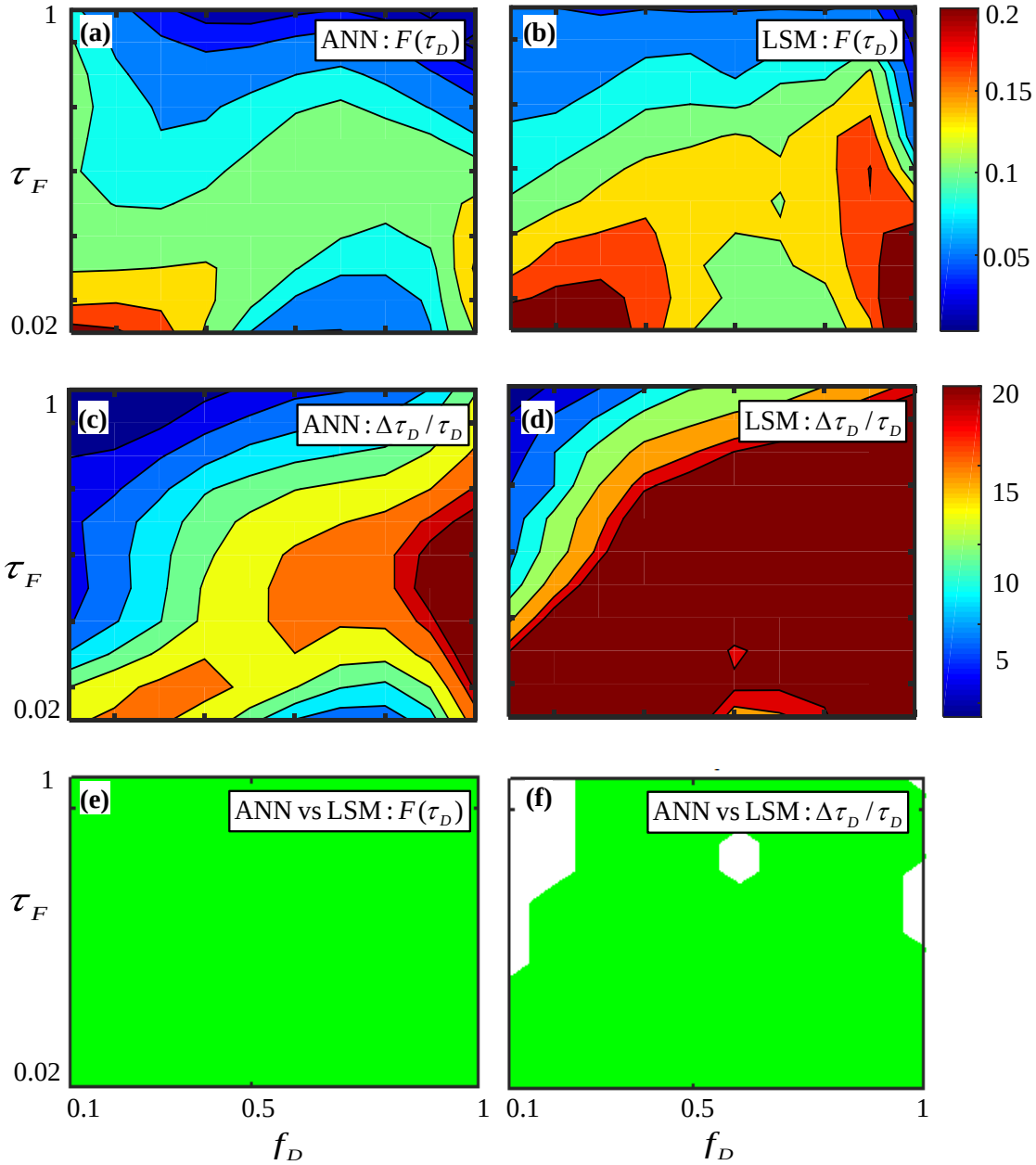

Figure 5.5 - Performances of τ_D calculated by ANN and LSM: (a) and (b) F-value; (c) and (d) the bias; (e) areas where ANN has better precision performance (green areas); (f) areas where ANN has better accuracy performance (green areas). For both precision and accuracy performance, the smaller result means the better performance. Figures 5.5(a), (b) and (e) show that ANN can provide better precision performance for all f_D. Figures 5.5(c), (d) and (f) show that ANN also produces less biased f_D.

Table 5.2 compares the processing time of ANN and LSM when using OpenMP CPU computation on a Windows PC (Intel (R) Xeon(R) E5-2609 v2 processor with 32GB

memory) on FLIM images of size 256×256. The proposed ANN-FLIM is a staggering *83-fold* faster than LSM.

Table 5.2 CPU processing time of ANN and LSM for synthesized data

Algorithms	**Image Size**	**Time** (s)	**ANN Speedup** (times)
ANN	256×256	0.86	83
LSM		71.7	

In terms of GPU acceleration, both algorithms have been tested on a NVIDIA Tesla K40 GPU device. Table 5.3 demonstrates the results of GPU acceleration for the same data used for Table 6.6.

Table 5.3 GPU acceleration of ANN and LSM for synthesized data

Algorithms	**Image Size**	**GPU Time** (s)	**ANN Speedup** (times)
ANN	256×256	0.038	141
LSM		0.538	

5.3.2 ANN-FLIM for IRF based fitting

To evaluate the proposed ANN-FLIM method, it was tested on synthesized data, comparing with LSIR and LSD-LE. More specifically, LSIR and LSD-LE used Levenberg-Marquardt based nonlinear least-squares runtime functions [54]. For this simulation, the bin width was h = 32ps, and the number of time bins was m = 165. As shown in figure 5.6, the normalized IRF was simulated as it can be measured from the FLIM system, where the peak appears at the 41st time bin and the full width at half maximum (FWHM) = 12h = 384ps.

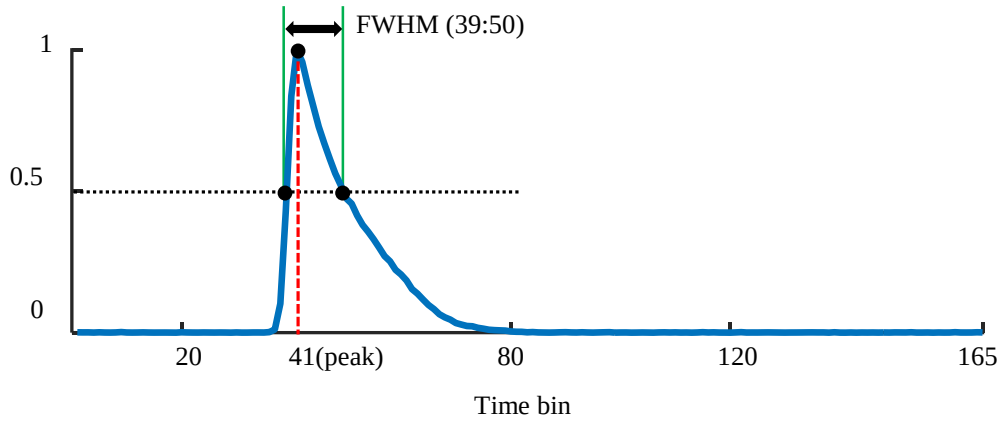

Figure 5.6 - Simulated IRF based on measured data. The green lines demarcate the full width of half-maximum, which spans time bin 39 to 50.

Table 5.4 specifies the range of each FLIM parameter. All the training samples were generated based on the parameters within the corresponding ranges as the ANN-FLIM for tail-fitting. For this study, we chose 7000 α ($\alpha = [K, f_D, \tau_F, \tau_D]$) vectors using the LDS method, and generated 120 histograms by adding Poisson noise on theoretical histogram for each vector. In total, we created 840,000 training samples for ANN training. Also, as demonstrated in chapter 4, the first 38 bins were disregarded in this case. After less than 3 hours of training (accelerated by a NVIDIA Tesla K40 GPU), the ANN was able to conduct the FLIM analysis directly.

Table 5.4 Simulation setup of ANN-FLIM

	K	f_D	τ_F (ns)	τ_D (ns)
Range	[9, 800]	[0, 1]	[0.024, 0.2]	[0.32, 2]

During the Monte-Carlo simulation, we generated 1000 histograms by adding Poisson noise for each α vector. To make a comprehensive evaluation, we compared the ANN method with LSIR under three different photon counts (i.e., 500, 1500 and 2500 counts), as shown in figures 5.7, 5.8, and 5.9 respectively. More specifically, the results are based on α vectors by varying the parameters (f_D = [0.2, 0.5, 0.9], τ_F = [0.05ns, 0.1ns, 0.15ns], τ_D = [0.5ns, 1ns, 1.5ns]). For each α vector, we generated 1000 histograms by adding Poisson noise. Figure 5.7 compares the accuracy and precision of the results generated by the ANN-FLIM and LSIR, where each histogram had around 500 total photon counts. Figures 5.7(a) and 5.7(c) show the bias of f_D and τ_F with the ground truth (τ_D = 1ns). The

bias of τ_D is demonstrated when τ_F is 0.1ns, as shown in figure 5.7(e). With the same configurations, Figs. 5.7(b), 5.7(d), and 5.7(f) present the precision of the two methods (F-values of f_D, τ_F, and τ_D). The results of ANN are given in red curves, and it is obvious that the ANN can provide comparable or even better outcomes (except when N_C is low and f_D is small). Although, when f_D is close to 0, the bias of f_D generated by the ANN is higher than the LSIR, the ANN is able to provide similar results when f_D is getting larger. As for the bias of τ_F, or τ_D, except when f_D is very small, both methods can provide satisfactory outputs. Moreover, it is clear that when these two methods have the similar bias performance, the performance of ANN-FLIM in terms of precision is better for ANN-FLIM**.** Figures 5.8 and 5.9 (N_C = 1500 and 2500 counts respectively) confirm that the ANN has potential to outperform LSIR. Besides its superior precision performance over LSIR, the bias of f_D is significantly improved when the photon counts increase. Overall, the ANN and LSIR have different optimized bias performances, whereas the ANN can always provide better precision.

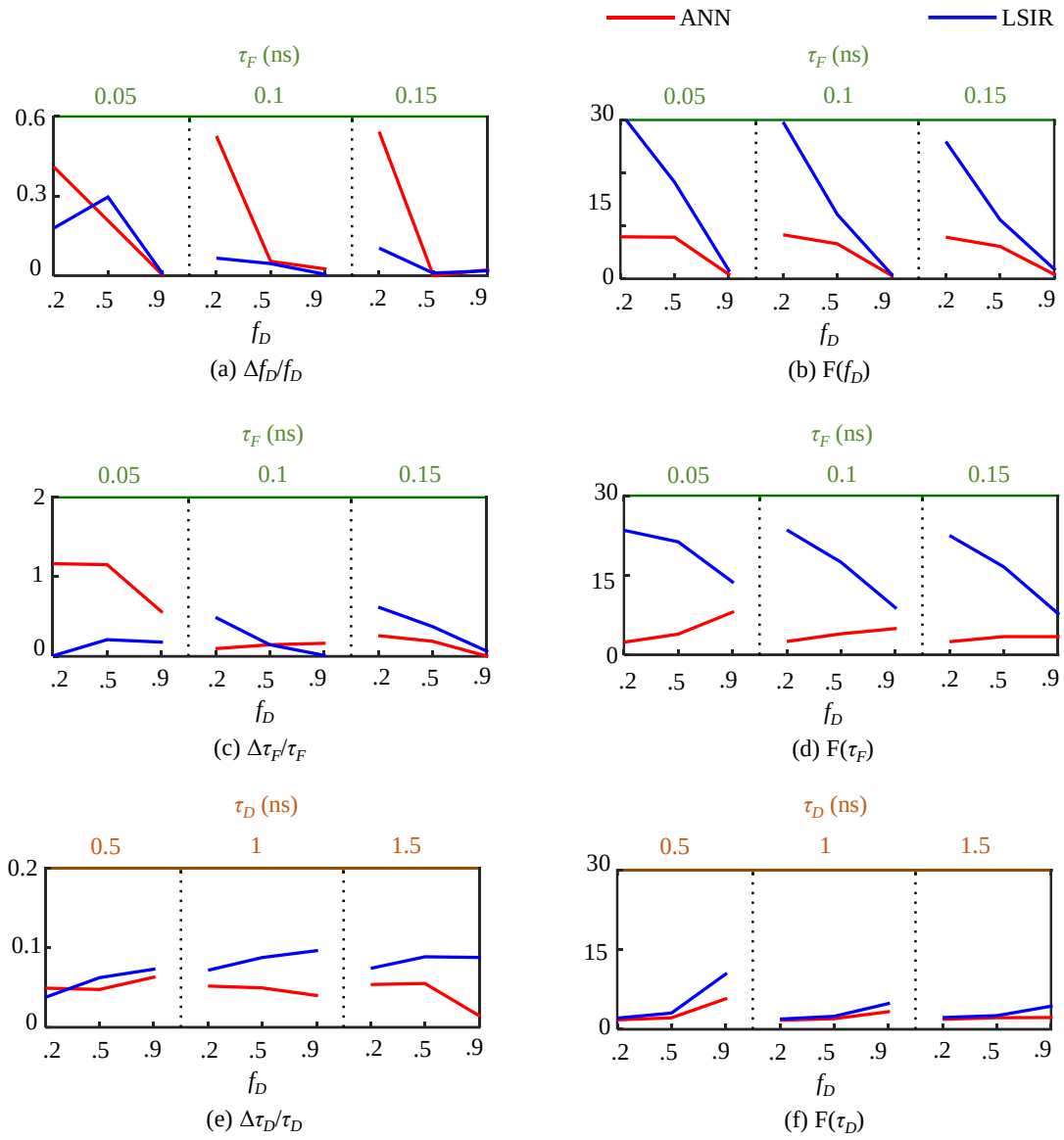

Figure 5.7 - ANN and LSIR performance comparison (500 counts): F-value of (a) $f_{D,}$, (c) τ_F, and (e) τ_D; the bias of (b)) $f_{D,}$, (d) τ_F, and (f) τ_D. The results of ANN are given in red curves, and it is obvious that the ANN can provide comparable or even better outcomes (except when N_C is low and f_D is small). Although, when f_D is close to 0, the bias of f_D generated by the ANN is higher than the LSIR, the ANN is able to provide similar results when f_D is getting larger. As for the bias of τ_F, or τ_D, except when f_D is very small, both methods can provide satisfactory outputs. Moreover, it is clear that when these two methods have the similar bias performance, the performance of ANN-FLIM in terms of precision is better for ANN-FLIM.

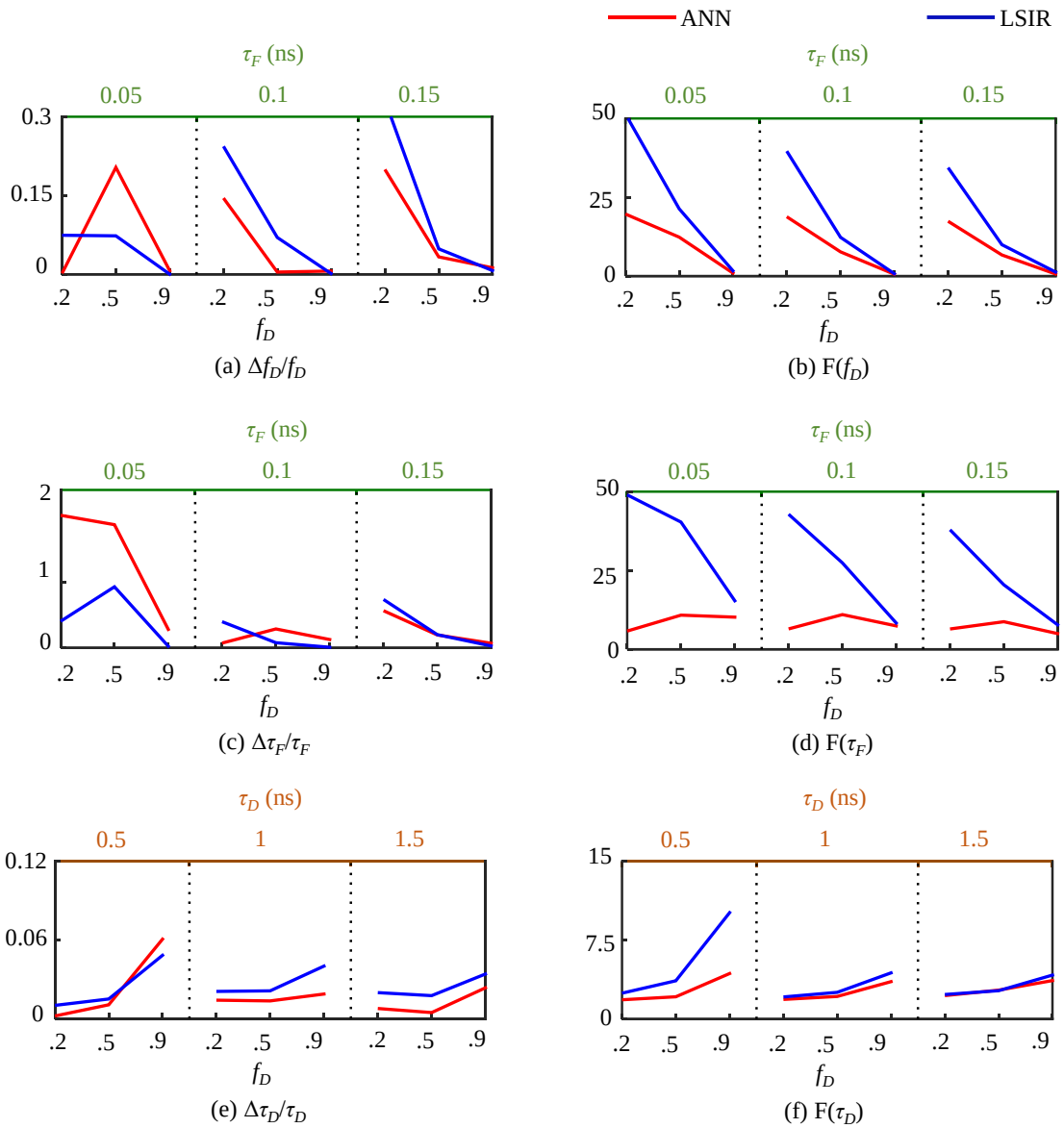

Figure 5.8 - ANN and LSIR performance comparison (1500 counts): F-value of (a) $f_{D,}$, (c) τ_F, and (e) τ_D; the bias of (b)) $f_{D,}$, (d) τ_F, and (f) τ_D. ANN has potential to outperform LSIR. Besides its superior precision performance over LSIR, the bias of f_D is significantly improved when the photon counts increase.

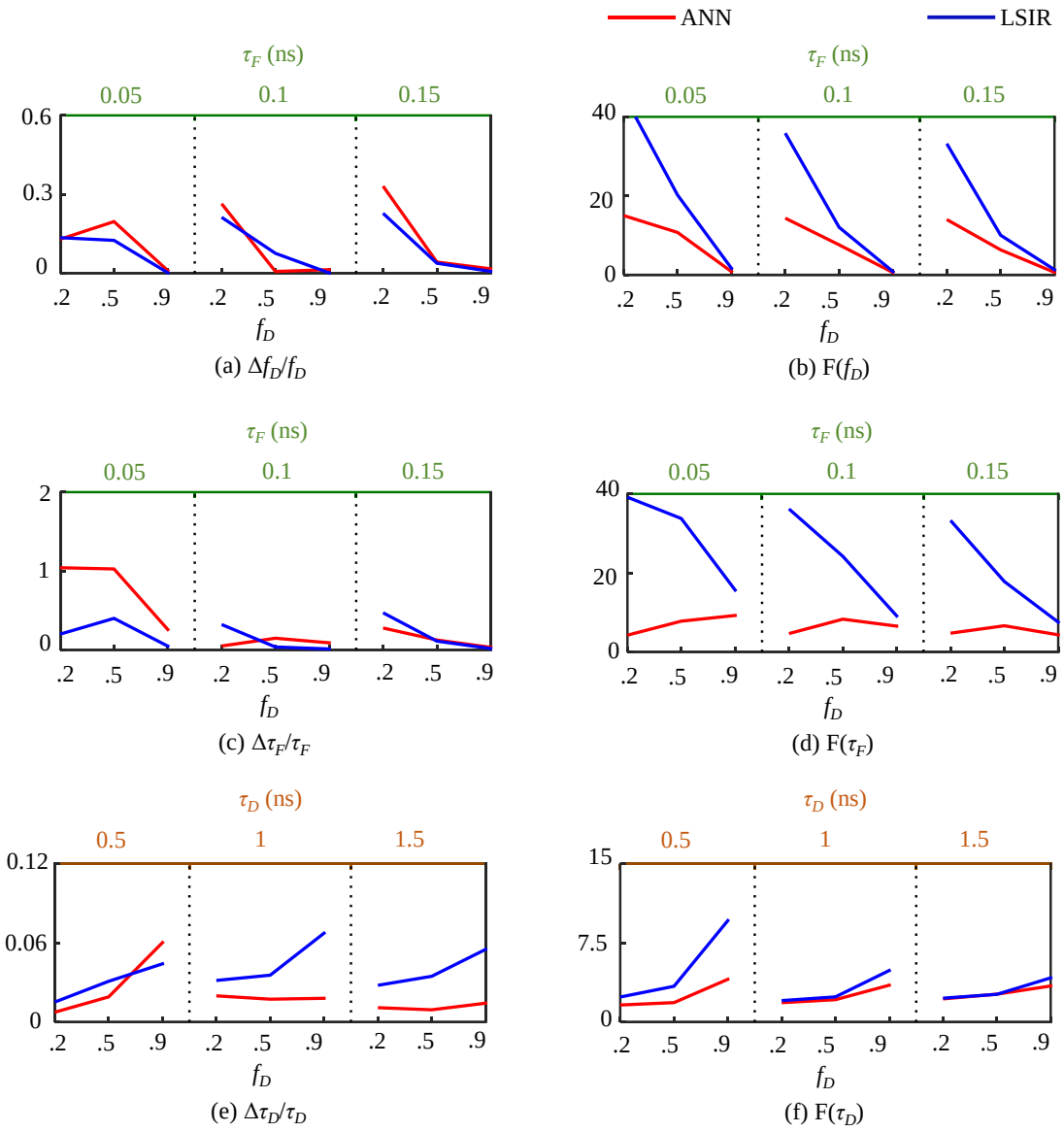

Figure 5.9 - ANN and LSIR performance comparison (2500 counts): F-value of (a) f_D,, (c) τ_F, and (e) τ_D; the bias of (b)) f_D,, (d) τ_F, and (f) τ_D. ANN has potential to outperform LSIR. Besides its superior precision performance over LSIR, the bias of f_D is significantly improved when the photon counts increase.

A unique feature of the proposed ANN-FLIM is that it is able to provide high-speed or even real-time FLIM analysis, making it an efficient tool to make timely analysis. Table 5.5 shows the analysis time for synthesized FLIM data based on OpenMP-CPU simulations. More specifically, LSIR used least squares method with Levenberg–Marquardt algorithm, and non-negative least squares method and least squares method with Levenberg–Marquardt algorithm were used for deconvolution and FLIM parameters

extraction of LSD-LE respectively. It clearly shows that the ANN-FLIM is more than 150-fold faster than other traditional approaches, dramatically reducing the processing time and making real-time FLIM analysis possible. In addition, table 5.6 demonstrates the processing time of GPU accelerations.

Table 5.5 Processing time of ANN, LSIR and LSD-LE for synthesized data

Algorithms	**Histogram Number**	**Time** (s)	**ANN Speedup** (times)
ANN		0.17	/
LSD-LE	9000	26.1	154
LSIR		76.8	452

Table 5.6 GPU acceleration of ANN, LSIR and LSD-LE for synthesized data

Algorithms	**Histogram Number**	**GPU Time** (s)	**ANN Speedup** (times)
ANN		0.0081	/
LSD-LE	9000	6.363	785
LSIR		7.288	1033

5.4 ANN-FLIM assessment on experimental data

The performances of ANN-FLIMs were also evaluated on real experimental data against LSM and LSIR. FLIM experiments were performed on daisy pollens and the data used in this thesis was measured by using a MicroTime 200 time-resolved confocal fluorescence microscope (PicoQuant, Germany). More specifically, the MicroTime 200 was equipped with the standard piezo scanner (Physik Instrumente; 100x100µm scan range) and a SPAD (SPCM-AQRH from Excelitas). The excitation source was a ps-pulsed diode laser (LDH-D-C-485) operating at 485nm with the pulse frequency of 20MHz (50ns for the TCSPC dynamic range), which was controlled by the PDL 828 "Sepia II" laser driver. The data was acquired by the HydraHarp 400 (bin width set to 8ps). Also, the FLIM image size contained 400 by 400 pixels.

5.4.1 ANN-FLIM for tail-fitting

We combined 40 original time bins together to generate a new histogram (new bin width is 8*40=320ps). The original data has more than 6000 time bins, which are not necessary and also delay lifetime calculation, and the proposed ANN-FLIM only requires 64 inputs. Since the original data included the influence of the IRF, for tail-fitting methods, the effective data should start at the peak of the histogram. To determine the overall peak (the time bin with highest photon count), we combined the data of all pixels together and created a single histogram. The overall peak is then chosen as the peak of this histogram. Finally, for each pixel, effective data started at this overall peak and ended at the following 63rd time bin (i.e., 64 time bins are used for lifetime calculation). The intensity image of daisy pollens is shown in figure 5.10 (a). Figures 5.10 (b)-(g) compare the τ_F and average lifetime τ_{Average} [$\tau_{\text{Average}} = f_D \cdot \tau_F + (1- f_D) \cdot \tau_D$] maps for ANN and LSM. From these images, it is easy to see that ANN is capable of extracting the features of the sample, and it shows similar results to those of LSM, especially for the images of average lifetime. On the other hand, comparing figures 5.10 (b) and (c) shows that in some pixels LSM failed to converge to correct estimations (dark red spots), which is due to its high sensitivity to initial conditions (to improve it might require quick estimations on f_D, τ_F and τ_D taking more analysis time), whereas ANN provides a superior success rate. To be specific, the success rate is 99.93% for ANN, and 95.93% for LSM. Figures 5.10 (d) and (e) show similar merged images (intensity and τ_{Average}) for ANN and LSM, respectively. Figures 5.11 (a) and (b) also shows that ANN produces similar τ_F , τ_{Average} and f_D histograms with LSM, except that LSM has more invalid pixels around $\tau_F \sim 0$ns and there is a slight difference in τ_D. The difference in τ_D is likely due to the different bias behaviours when f_D is closer to 1 (see figure 5.11 (b)) and $\tau_D > 3$ns. However, LSM and ANN still show similar τ_{Average} histograms. From these results, one can learn that for Daisy Pollen the shorter lifetime is more likely to appear at the edge of the cell, whereas the longer lifetime is distributed more homogeneously.

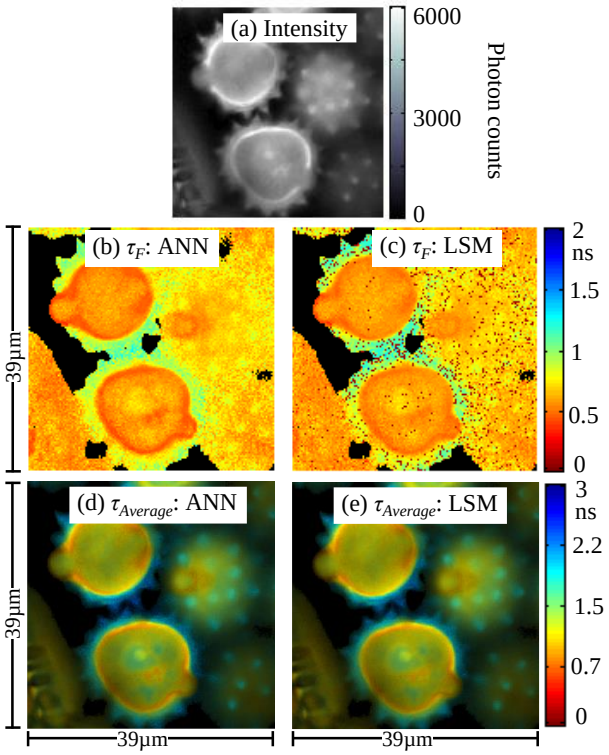

Figure 5.10 - (a) Intensity image, (b) ANN and (c) LSM τ_F images, (d) ANN and (e) LSM merged intensity and $\tau_{Average}$ images (Reproduced from [116]). ANN is capable of extracting the features of the sample, and it shows similar results to those of LSM, especially for the images of average lifetime.

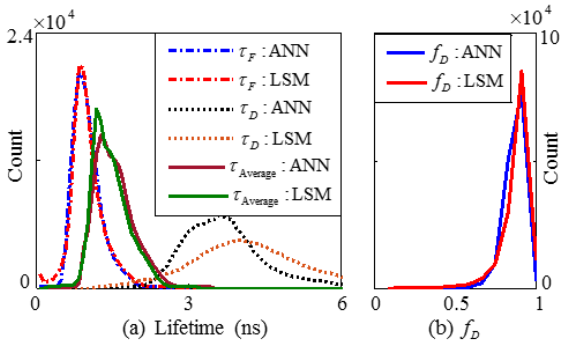

Figure 5.11 - (a) Lifetime and (b) f_D histograms of the experimental data (Reproduced from [116]). ANN produces similar τ_F , $\tau_{Average}$ and f_D histograms with LSM, except that LSM has more invalid pixels around $\tau_F \sim 0$ns and there is a slight difference in τ_D.

A more convincing feature of ANN is demonstrated in table 5.7 and table 5.8, where ANN is at least 75-fold faster than LSM. Together with the observation from the previous analysis on synthesized data, the speed of LSM analysis is subject to the choice of initial conditions, whereas ANN does not require any initial conditions.

Table 5.7 CPU processing time of ANN and LSM for experimental data

Algorithms	**Image Size**	**Time** (s)	**ANN Speedup** (times)
ANN	400×400	1.8	120
LSM		216.8	

Table 5.8 GPU processing time of ANN and LSM for experimental data

Algorithms	**Image Size**	**Time** (s)	**ANN Speedup** (times)
ANN	400×400	0.14	75
LSM		10.5	

5.4.2 ANN-FLIM for IRF based fitting

Taking a different approach from tail-fitting here, we combined 4 original time bins together to generate a new histogram (new bin width is 8*4=32ps). Since the original data has more than 6000 time bins, which is not necessary and also delay lifetime calculation, and the proposed ANN-FLIM only requires 128 inputs. With the same experimental setup, the IRF was recorded by removing the background noise of the original experimental data, as shown in figure 5.12. In order to increase the efficiency of data processing, the same as for simulation, the first 37 time bins which make little contribution on the lifetime calculation are neglected. In this case, 128 bins (form 38^{th} bin to 165^{th} bin) are considered for lifetime calculation. After obtaining the FLIM parameters, the FLIM images were generated, as shown in figure 5.13. In this case, the photon count per pixel ranges from 0 to 3000, and the intensity image is shown in figure 5.13 (a). As shown in figures 5.13 (b) and (c), the two algorithms generate similar merged FLIM images, i.e., combining average lifetime images, $\tau_{Average} = f_D\tau_F + (1- f_D)\tau_D$, with intensity image, respectively. Whilst, the lifetime image generated by LSIR is noisier however, due to different bias performances, these two methods produce different τ_D images. To further compare these two methods, lifetime and f_D histograms of the experimental data have been shown in figures 5.14 (a) and (b) respectively. From these figures, ANN-FLIM can have similar performances as LSIR, except the bias of τ_D, and the results generated by ANN-FLIM have higher precision. As one can find from these histograms, the ANN method provides higher precision for both τ_F and f_D, which is consistent with the

simulation results. Moreover, the histograms help us to understand the cause of the gap of τ_D (0.4ns distance from peak to peak as shown in figure 5.14 (a)), from where one can learn that f_D is close to 1. Examining the simulations, when f_D is closer to 1, the accuracy of τ_D results decrease for both methods, and the LSIR has higher bias than the counterpart. More specifically, the ANN produces τ_D with mean value above the ground truth, whereas the LSIR favors the mean value below the true value. Although the bias of τ_D between these two algorithms is not negligible, lifetime images with similar contrast can be acquired when the overall bias is eliminated, e.g. generating τ_D image of the ANN method with the color bar range from 0.9ns to 3.1ns as shown in figure 5.15. For FLIM analysis, since the key is to generate fluorescence lifetime contrast images, FLIM images with overall bias are acceptable.

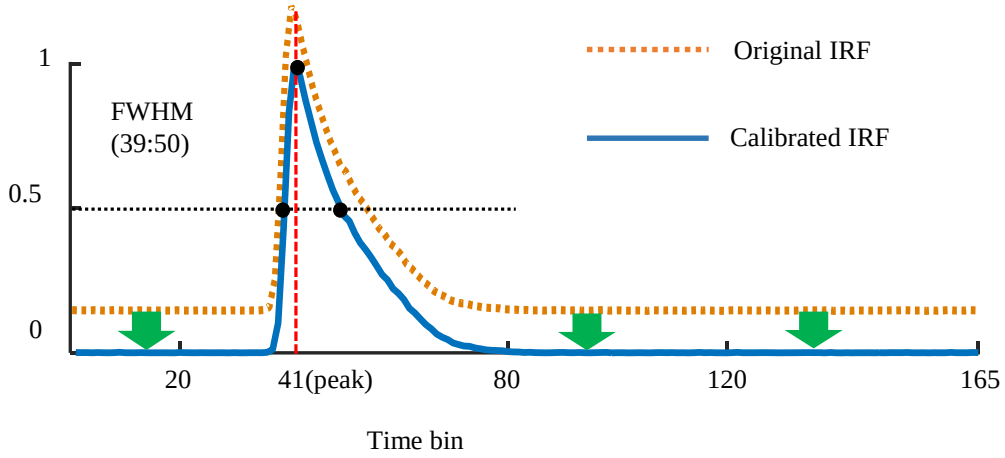

Figure 5.12 - Background noise calibration of the experimental IRF.

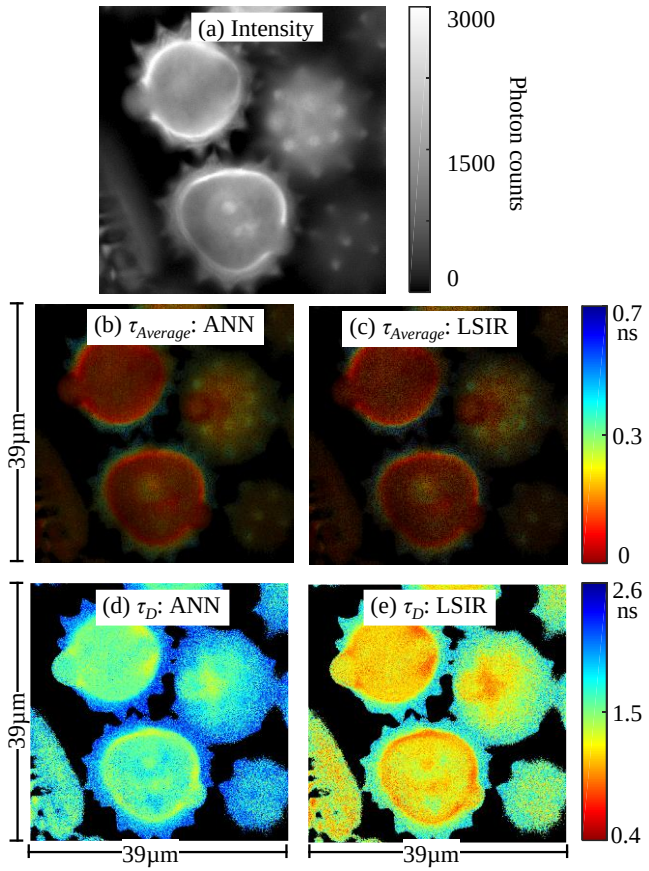

Figure 5.13 - (a) Intensity image, (b) ANN and (c) LSIR τ_{Average} images combined with intensity. (d) ANN and (e) LSIR τ_D images. Two algorithms generate similar merged FLIM images. Whilst, the lifetime image generated by LSIR is noisier. However, due to different bias performances, these two methods produce different τ_D images.

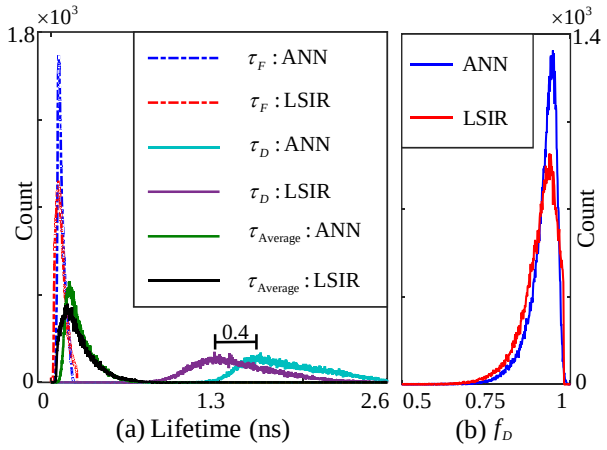

Figure 5.14 - (a) Lifetime and (b) f_D histograms of the experimental data. ANN-FLIM can have similar performances as LSIR, except the bias of τ_D, and the results generated by ANN-FLIM have higher precision.

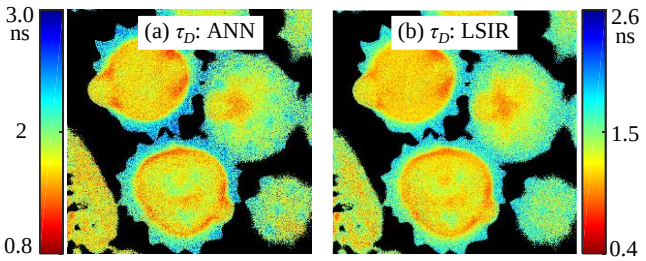

Figure 5.15 - (a) New ANN τ_D image, (b) LSIR τ_D image. Although the bias of τ_D between these two algorithms is unneglectable, lifetime images with similar contrast can be acquired when the overall bias is eliminated.

Moreover, in terms of the processing time, the proposed ANN method has an unbeatable advantage. As demonstrated in table 5.9, both LSD-LE and LSIR are far slower than ANN-FLIM. Although LSD-LE is more than 3-fold faster than LSIR, it has two time consuming steps (i.e., deconvolution and lifetime calculation). The ANN method however calculates the lifetimes directly, which allows it to generate the outputs 1449-fold faster than LSD-LE. Even though high-speed simple algorithms for bi-exponential decay analysis have been introduced (e.g., BCMM [45]), due to the existence of the deconvolution process, the proposed method is still at least 515-fold faster.

Table 5.9 CPU processing time of ANN and LSM for experimental data

Algorithms	**Image Size**	**Time** (s)			**ANN Speedup** (times)
		DE-CON	**CALC**	**Overall**	
ANN		/	1.2	1.2	/
LSD-LE	400×400	189.4	145.4	334.8	279
LSIR		/	819.6	819.6	683

DE-CON: deconvolution; CALC: lifetime calculation.

Table 5.10 GPU processing time of ANN and LSM for experimental data

Algorithms	**Image Size**	**Time** (s)			**ANN Speedup** (times)
		DE-CON	**CALC**	**Overall**	
ANN		/	0.12	0.12	/
LSD-LE	400×400	44.5	9.6	54.1	451
LSIR		/	56.2	56.2	468

DE-CON: deconvolution; CALC: lifetime calculation.

5.5 Summary

This chapter begins with the GPU acceleration of ANN-FLIM methods. Since the fundamental operation in an ANN is the inner-product between an input vector and a weight vector in each layer, the GPU implementation of ANN can be represented by CUDA matrix multiplication. Then, ANN-FLIM methods (both for tail-fitting and IRF based fitting) have been evaluated on synthesized data. By comparing with standard iterative algorithms, we can find that the proposed ANN-FLIM methods are able to provide comparable results (even better results in some cases). In terms of calculation speed, ANN-FLIM methods are much faster (more than 400-fold faster) than their counterparts. Similar results have also been found from experimental data evaluation. With the help of GPUs, ANN-FLIM analysis has achieved at least 10× speedup over the CPU-only computation. ANN-FLIM methods with GPU acceleration are capable of delivering real-time FLIM analysis.

Chapter 6 Discussion and Conclusion

6.1 Overview

The design of a high-speed FLIM analysis system is faced with a huge data problem, which continues to grow massively, with the development of high-speed FLIM data acquisition systems and the increasing demand of high-resolution FLIM images. To tackle the problems, FLIM developers have to either develop more efficient algorithms, such as the simple algorithms that are mentioned in chapter 2, or make use of more powerful computational hardware. This study is dedicated to developing a high-speed analysis system by harvesting the advantage of each FLIM algorithm as well as the outstanding computational performance of GPUs.

Different types of algorithms have their own advantages and disadvantages, and there is no single algorithm that can be considered as the best one or one that can replace all others. It is therefore necessary to understand how to choose an appropriate method for each specific application.

This chapter first presents a summary of the final GPU accelerated FLIM analysis system. Then, a discussion over the features and limitations of each type of algorithms are demonstrated, as well as some further understanding of them. Additionally, some considerations of GPU acceleration for FLIM analysis are demonstrated.

This chapter ends by giving the conclusions of this research and some recommendations for future work.

6.2 Architecture of GPU accelerated FLIM analysis system

The GPU accelerated FLIM system aims to combine a variety of FLIM algorithms and provide comprehensive analysis solution at a high-speed manner for different applications. Figure 6.1 illustrates the structure of the GPU enhanced FLIM analysis system. Before lifetime calculation, it is important to decide the model of exponential decay (e.g., single-exponential decay, bi-exponential decay), and usually this can be done by users based on their experience. Then, depending on applications and requirements, a most appropriate FLIM algorithm can be chosen based on the features of each algorithm. After system setup (e.g., number of time bins, bin width), FLIM data will be transferred to GPU devices for lifetime calculations. Generated FLIM results for each pixel are then delivered back to the CPU. With a predefined colour map (can be changed manually), the image generator is able to determine the colour for each pixel and the whole FLIM image can be acquired accordingly. In addition, if users are not sure which algorithm works best, they can select different algorithms and determine the final algorithm by observing the FLIM images.

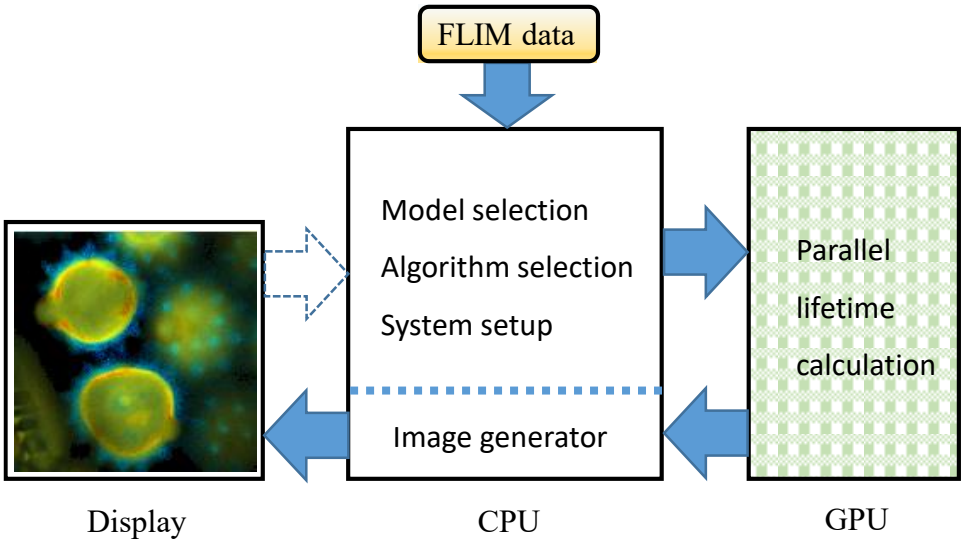

Figure 6.1 - The architecture of GPU accelerated FLIM analysis system. First, we have to decide the model of exponential decay and select appropriate algorithm for lifetime analysis. After finishing system setup, FLIM data will be transferred to GPU devices for lifetime calculations. Finally, image generator is able to generate FLIM images based on the results from GPU. In addition, if users are not sure which algorithm works best, they can select different algorithms and determine the final algorithm by observing the FLIM images.

6.2.1 Algorithm selection of FLIM analysis

It is true that there is no algorithm that can replace all other algorithms for all possible applications. Instead, based on the specific requirements for each application, we can choose a better algorithm to complete the work. A simple diagram of algorithm selection is shown in figure 6.2. First, we have to decide which fitting model to follow: tail-fitting or IRF based fitting. Then, a final algorithm under selected model can be determined based on the features of each algorithm.

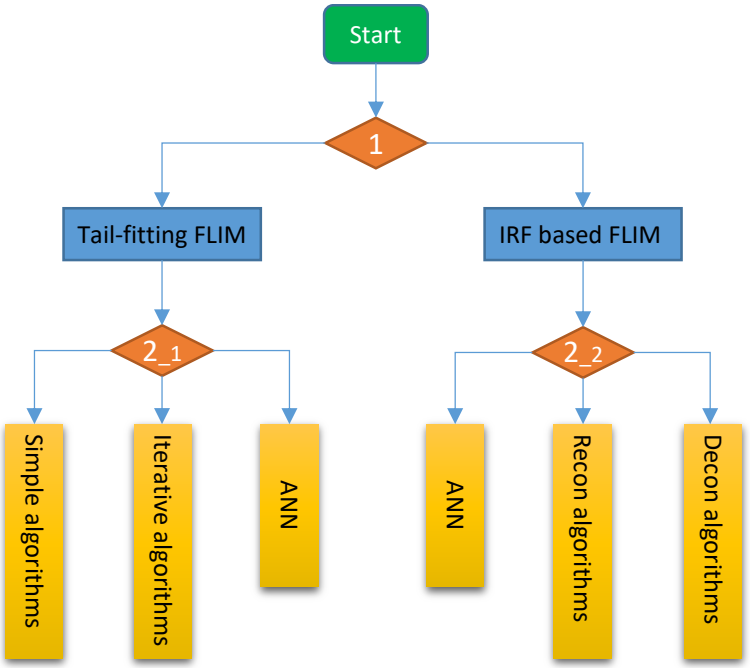

Figure 6.2 - A simple diagram of algorithm selection for FLIM analysis. First, to decide which fitting model to follow: tail-fitting or IRF based fitting. Then, final algorithm under selected model can be determined based on the features of each algorithm.

1) Tail-fitting and IRF based fitting

For FLIM analysis, there are two common methods to calculate lifetime parameters: tail-fitting methods [114, 128-130] and IRF based fitting methods [47, 49, 51, 131]. It can be difficult to accurately determine IRF for a FLIM system, so tail-fitting methods are still commonly used. Moreover, tail-fitting methods are normally faster than IRF based fitting methods, as they use a simpler fitting model. Tail-fitting methods are sufficient for analysing mono-exponential decay data [132]. As for bi-exponential decay, it is possible

to recover the two lifetimes, but the results depend strongly on the region that is chosen for fitting [128]. The long lifetime component determined by tail-fitting methods are more reliable than the short component, and the short component detected has higher bias than that is achieved by IRF based fitting methods [133]. However, lifetimes determined by IRF based fitting procedures are generally considered to be more reliable than tail-fitting methods [128, 133]. Also, for some applications, it is necessary to determine the short lifetime, so IRF based fitting procedures have to be applied.

2) Feature comparison of FLIM algorithms

A specific FLIM algorithm has to been chosen, after deciding the fitting method (i.e., tail-fitting or IRF based fitting). Criteria for how to choose an algorithm can vary for each application. Table 6.1 demonstrates the features of each commonly used algorithm, as well as ANN-FLIM methods.

Table 6.1 Feature of FLIM algorithms.

Algorithms	**Fitting method**	**Decay model**	**Performance ranking**	**Speed ranking**
IEM	Tail-fitting	Single	1-4	1-1
CMM	Tail-fitting	Single	1-3	1-2
PM	Tail-fitting	Double	2-5	2-1
BCMM	Tail-fitting	Double	2-3	2-2
LSM	Tail-fitting	Single/ Double	1-2 2-2	1-3 2-4
GA	Tail-fitting	Single/ Double	1-1 2-1	1-4 2-5
LSIR	IRF based	Single/ Double	3-1	3-3
LSD-LE	IRF based	Single/ Double	3-2	3-2
ANN	Tail-fitting/ IRF based	Single/ Double	2-4 3-3	2-3 3-1

"*i*-*j*" represents the *j*th best algorithm in the *i*th category (three categories: single exponential decay for tail-fitting, double exponential decay for tail-fitting, and IRF based FLIM analysis).

6.2.2 Analysis mode

It is always better to pursue high-speed FLIM analysis, although to some extent a trade-off can be made between high performance (i.e., accuracy and precision) and high speed. The proposed FLIM analysis system can work in three modes: offline, online, and offline enhanced online analysis. Some applications require comprehensive analysis based on the data acquired from a target sample, and relatively slow analysis can be acceptable. Highly accurate results are obtained from historical data with the help of high performance FLIM algorithms. For these applications offline analysis mode is the best choice. However many applications require real-time analysis to capture dynamic cellular activities, so it is necessary to choose a high-speed algorithm for data analysis. Online mode means generating FLIM images as quickly as possible. In addition, for online analysis, offline enhancement is helpful when researchers identify a specific time line or area in the FLIM image that is important.

6.3 Standard methods

As mentioned in Chapter 2, there are a variety of standard FLIM algorithms, which have covered every scenario of FLIM analysis (i.e., both tail-fitting and IRF based fitting applications). Most of them have been widely used, and are still fuelling the development of FLIM systems.

6.3.1 Features

Figures 3.23, 3.24, 5.3-5.5, and 5.7-5.9 have demonstrated that, in most cases, standard algorithms are capable of delivering satisfactory results with high performance. From figures 3.23 and 3.24, one can easily find that the generated FLIM images are very close to the theoretical images. The performance of standard algorithms can be further verified by the experimental results as demonstrated in section 5.4 as well as in [40, 45, 85, 134, 135]. Different kinds of experimental data (e.g., the interaction between Ras-related C3 botulinum toxin substrate 1 and the p21-activated kinase in COS-7 cells, Cy5-ssDNA-

GNRs (gold nanorods) labeled Hek293 cells, and live MCF-7 breast cancer cells) have been used to test the FLIM systems. More specifically, traditional iterative algorithms (such as LSM, GA, LSIR, and LSD-LE) are more likely to produce FLIM results with higher accuracy. Especially, for tail-fitting applications, GA is able to provide much better solution, as shown in figure 3.24.

Standard algorithms, especially iterative algorithms, are capable of delivering reliable results. Figures 3.23 and 3.24 demonstrate that iterative algorithms (e.g., LSM and GA) as well as some simple algorithms (e.g., IEM, CMM, PM, and BCMM) can generate FLIM results with high precision, for example, for single-exponential decays the averaged relative precision values of CMM and LSM are 0.026 and 0.03, and for bi-exponential decays the averaged relative precision values of PM, BCMM, and GA are 0.048, 0.046, and 0.013, respectively. Moreover, iterative algorithms and CMM have wide optimal working range, which means reliable results can been acquired under a variety of circumstances. To be more specific, the performance of FLIM algorithms is influenced by the number of photons, the measurement window, the proportions of lifetime components, and the ratio between lifetime components. As one can learn from figures 5.7 to 5.9, even though the total photon count is low, the accuracy of the FLIM results generated by LSIR are similar when the photon count is high. And from [29], both iterative algorithms and CMM have a wide optimal measurement window, where algorithms can generate lifetime results with high accuracy and precision.

In terms of calculation speed, all of the simple algorithms have an unbeatable advantage. Compared with iterative algorithms, simple algorithms are at least 50-fold faster. With the help of GPUs, real-time tail-fitting FLIM analysis can be easily achieved (the image analysis frame rates are around 40 fps). This advantage is especially valuable for *in vivo* measurements, e.g., to detect protein-protein interactions with FRET.

6.3.2 Limitations

Although simple algorithms are overwhelmingly faster than iterative algorithms, most of them have lower accuracy and precision performance and reliability. For example, as

illustrated in figure 3.23(b), the averaged relative precision value of IEM is 0.15, which is 5-fold higher than LSM. Moreover, [29] has also indicated that IEM has narrower optimal windows. These disadvantages remind us that it is necessary to carefully select the best algorithms for every application based on the specific detail of the experiments.

Iterative algorithms, on the other hand, have the one obvious disadvantage that lifetime calculation is massively time-consuming for real–time FLIM analysis. Even though parallel high-performance processors have been used, the image analysis frame rate (e.g., it is 0.85 for LSM for single-exponential decay analysis) is far below that is require by real-time imaging.

6.4 ANN-FLIM methods

The results presented in the previous chapter have demonstrated the potentials of proposed ANN-FLIM methods for both tail-fitting and IRF based fitting.

6.4.1 Features

First of all, ANN-FLIM methods are novel high-speed FLIM analysis tool, where ANNs have been successfully introduced into this area for the first time. Feedforward neural networks, used in this research, are one of the most common and the simplest types of ANN. The proposed ANN-FLIM methods does not require iterative searching procedures. More specifically, during the lifetime calculation, only a small amount of simple operations (e.g., summation, subtraction, division and exponential operation) are conducted. This feature allows them to achieve at least 100× speedup compared with traditional iterative algorithms, as demonstrated in tables 5.3 and 5.6. With GPU acceleration, it is suggested that ANN-FLIM methods are able to deliver real-time FLIM analysis (i.e., it only takes ANN-FLIM method around 60ms to generate a 256×256 FLIM image, which means the image analysis frame rate is around 17 fps).

ANN-FLIM methods do not require initial conditions, which are necessary for traditional iterative algorithms. Taking LSM as an example, an optimization algorithm (e.g.,

Levenberg–Marquardt algorithm) is used to find better parameters to minimize the cost function. Since it finds only a local minimum, which is not necessarily the global minimum, the initial conditions or initial guess of target parameters may affect the outputs significantly. And for most cases, it is difficult to find the best initial conditions. In other words, for the same FLIM histogram, LSM with different initial conditions may lead to different lifetime results. Without an initial guess, ANN-FLIM methods can always generate the same results with the same histogram data, as simple algorithms do.

Although ANN-FLIM algorithms cannot replace traditional iterative algorithms completely, they can generate comparable or even better results in terms of accuracy and precision performance. From figures 5.3(c)-(d), 5.4(c)-(d) and 5.5(c)-(d), one can learn that LSM generates slightly better results with higher accuracy, but their estimations are on the same order. One can also find that ANN-FLIM algorithms and LSM have different optimized areas. Similar results can also be found in figures 5.7-5.9. Moreover, from F-value results as demonstrated in figures 5.3-5.5 and 5.7-5.9, it is obvious that for most cases ANN-FLIM algorithms are able to deliver better results.

As intelligent algorithms, the proposed ANN-FLIM algorithms are easier to be applied to different FLIM models than simple algorithms. For example, IEM cannot be adapted to double-exponential decay analysis, and simple algorithms are only for tail-fitting analysis. Instead, ANN-FLIM algorithms can be adapted to both single and double or even more complicated exponential decays without redesigning the entire algorithms. This can be done by changing the number of neurons and collecting new training data. Although the IRF makes FLIM analysis more difficult, with limited modifications and efforts, ANN-FLIM algorithms have been successfully adapted, as illustrated in section 4.4. The potential of ANN-FLIM algorithms can be further explored due to the dramatic development of artificial intelligent and high-performance computing technologies.

6.4.2 Limitations

In terms of calculation speed of FLIM algorithms, it is undeniable that simple algorithms are the best algorithms with fastest speed for single-exponential decay analysis in tail-

fitting approaches. Although ANN-FLIM methods are iteration-free and much faster than LSM and other iterative algorithms, they involve many inner product operations. So for the single-exponential decay analysis, there is no need to design an ANN-FLIM algorithm for it. Whereas, for more complicated applications, especially IRF based FLIM analysis, ANN-FLIM methods can be very beneficial. Additionally, traditional iterative algorithms and ANN-FLIM methods have different optimized areas in terms of accuracy and precision performance. This means that ANN-FLIM methods are not able to replace traditional iterative algorithms completely.

6.4.3 Further considerations

There are some further considerations about using ANNs for FLIM analysis, including recommendations about how to optimize the network.

6.4.3.1 Ranges of training samples

In order to maximize the performance of ANN, it is necessary to improve the generalization of the network. Here, we assume general approaches for improving generalization have been applied, e.g., retraining neural networks, multiple neural networks, early stopping, data division, regularization [101]. For FLIM analysis, more complicated network and more training samples are required to both improve the generalization and increase the performance of ANN-FLIM methods. These requirements dramatically increase the difficulty and decrease the speed of network training, and also slow down the speed of lifetime calculations. Figure 6.3 demonstrates three architectures of networks, i.e., NET1: two hidden layers (32, 64 neurons), NET2: two hidden layers (32, 128 neurons), NET3: four hidden layers (16, 16, 64, 64 neurons). More specifically, NET1 has less total neurons (96) in hidden layers, and NET2 and NET3 have the same number of total neurons (160) in hidden layers. Although NET2 and NET3 have the same number of neurons, NET3 has much less weight variables (3392) than NET2 (6272). Moreover, as demonstrated on the figures, last hidden layer (or last two hidden layers) are not fully connected. This is based on the knowledge that not all parameters are needed for FLIM analysis, and this configuration can speedup lifetime calculation by eliminating

unnecessary neurons. Table 6.2 compares the performance of networks with different architectures and training configurations. From the 'Total Training Time' column, one can see that more neurons or more training samples requires longer training time, e.g., for NET1, it takes 4857s to train with 210,000 samples, which is much faster than to train with 420,000 samples (43976s). We can learn from the 'Beyond-range Performance' column that networks can have better generalization performance when trained with samples covering wider ranges. The same column also shows that more training samples and more complicated networks do not necessarily mean better performance, because it is more difficult to train the network and the training technique used in this work is not sufficient. Moreover, it is clear that networks with more neurons or weights will slow the lifetime calculation process. By further examining the FLIM analysis, it is not difficult to find that overall generalization is not necessary. For example, although negative values of training targets can be generated, FLIM results can never practically be negative values. Moreover, in most cases, before carrying out the FLIM experiments, researchers can always find out the range of the FLIM parameters for the specific sample and data acquisition equipment. This pre-understanding of FLIM experiments can be very beneficial, and can be used to decrease the difficulties of ANN training and improve the performance of ANN-FLIM methods. A sub-generalization can be achieved by constraining the range of training targets (e.g., define the range of each parameter in vector $\alpha = [K, f_D, \tau_F, \tau_D]$ for bi-exponential decay), and a dedicated ANN can be trained for the given ranges. As demonstrated in the Table 6.2, this network does not perform well beyond the ranges, so it is better to define the proper range before training the network.

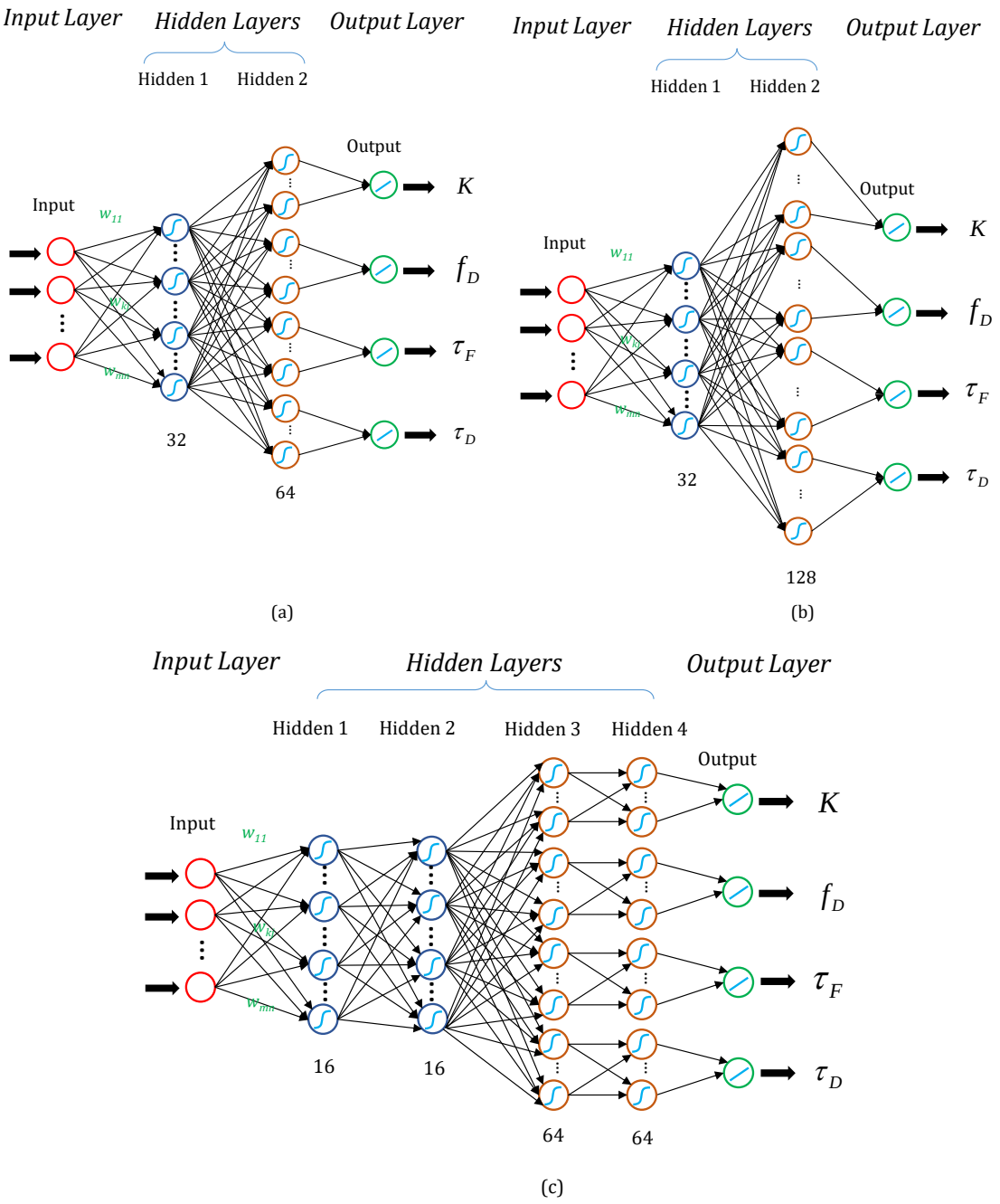

Figure 6.3 - Architectures of networks (a) NET1: two hidden layers (32, 64 neurons), (b) NET2: two hidden layers (32, 128 neurons), (c) NET3: four hidden layers (16, 16, 64, 64 neurons). More specifically, NET1 has less total neurons (96) in hidden layers, and NET2 and NET3 have the same number of total neurons (160) in hidden layers. Although NET2 and NET3 have the same number of neurons, NET3 has much less weight variables (3392) than NET2 (6272). Moreover, as demonstrated on the figures, last hidden layer (or last two hidden layers) are not fully connected. This is based on the knowledge that not all parameters are need for FLIM analysis, and this configuration can speedup lifetime calculation by eliminating unnecessary neurons.

Table 6.2 Performance of networks with different architectures and training configurations

Architectures	**Parameter Range**	**Training Sample**	**Training Time at 20th Epoch**(s)	**Total Training Time** (s)	**Total Training Epoch**	**Calculation Time per 256×256 pixels** (s)	**Within-range Performance** [a]	**Beyond-range Performance** [b]
NET1			2430	4857	40	0.78	11.82	3.99
NET2	Narrow	210,000	4520	6536	29	0.93	11.39	4.02
NET3			1860	13230	143	0.83	22.75	3.83
NET1			2440	4386	36		8.25	2.55
NET2	Wide	210,000	4400	12638	58	/	8.63	2.44
NET3			1960	6338	65		13.05	7.95
NET1			5480	43976	161		9.18	2.64
NET2	Wide	420,000	19860	25900	26	/	7.96	2.54
NET3			4260	33694	160		13.22	8.01

[a] Calculation formula: 100 / (average error of τ_F, when target τ_F is within [0.02, 1]). [b] Calculation formula: 100 / (average error of τ_F, when target τ_F is beyond [0.02, 1]).

6.4.3.2 Architecture of ANNs

For ANNs, the more complicated the structure (or more neurons) the more powerful they can be. This rule can also be applied to FLIM analysis. However excessively complicated structures always lead to training problems, i.e., slowing down the training process or sometimes the networks cannot be well trained (as demonstrated in table 6.2). There are no certain rules about how to choose the structure or number of neurons. However, in order to make a quick calculation, MLP is chosen as the overall architecture for FLIM analysis. In terms of how to decide the number of hidden layers, it is recommended to start with one layer and gradually increase the number of neurons. For this research, where GPUs are used to accelerate the lifetime calculation, the number of neurons in each layer is suggested to be a multiple of 32 (but it is not necessary for every circumstance). This is because GPUs execute threads in warps (as described in chapter 3), and warp divergence can slow the computation. From practice, this configuration is also more convenient for CUDA programming. As demonstrated in section 4.3.2, for tail-fitting a network with 2 hidden layers has been proposed which is able to calculate lifetimes efficiently. While for IRF based FLIM analysis, the same network architecture (but with more neurons) is not able to work as required. Therefore, I increased the number of hidden layers, and the results (see figure 6.4) proved that the architecture demonstrated in section 4.4.2 is better (especially for τ_D). As one can see from figure 6.4, although the F-values are similar, the bias produced by 2-hidden-layer network is much higher. One more consideration of designing a network for FLIM analysis is that not all outputs have to be calculated when the network is used during lifetime calculation. For example, although there are four parameters in the vector α, K is usually not needed. So it is beneficial to use more than two hidden layers, and in the second hidden layer we can group the neurons into separated sets, as shown in the second hidden layer in figure 4.8. With this configuration, less computation will be involved and only calculations of sets corresponding to required targets are needed. Overall, this architecture allows the training process to improve the performance by considering all the targets, and also speeds up the lifetime calculation by avoiding unnecessary computations.

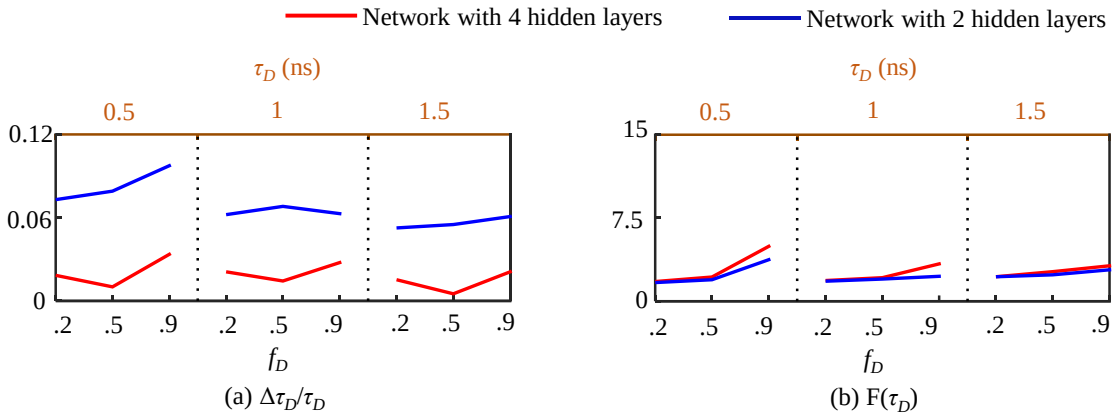

Figure 6.4 – Performance comparison of networks with different architectures: (a) the bias of τ_D. (b) F-value of τ_D. Although the F-values are similar, the bias produced by 2-hidden-layer network is much higher.

6.5 GPU acceleration

GPUs have dramatically increased the speed of lifetime calculations. This section describes the advantages of the GPU acceleration, as well as limitations.

6.5.1 Features

It is undeniable that GPUs can significantly accelerate parallel computing for more and more applications, including FLIM analysis. Currently, GPUs are able to achieve 10× to 100× or more speedup for certain applications, compared with CPUs. This outstanding achievement mainly arises from their floating-point performance and memory bandwidth. As one can find from figures 3.1 and 3.2 in chapter 3, both of them have evolved dramatically. Compared with CPUs, a state-of-the-art GPU is able to deliver a theoretical 11 Tera floating-point operations per second (TFLOPS), which is at least 7-fold higher than the latest CPU. A GPU also has much higher memory bandwidth than a CPU. These features have allowed GPUs to deliver a huge improvement for FLIM analysis. From the results presented in chapters 3 and 5 (e.g., tables 3.6, 3.7, 5.3 and 5.6), we can learn that for any FLIM algorithms (except LSD-LE) at least 10× speedups have been achieved.

In terms of the price per TFLOPS, GPUs are cheaper than CPUs. For example, the price of a NVIDIA Tesla K80 GPU (it can deliver 8.7 TFLOPS) is less than $5000, while the price of an Intel® Xeon® Processor E5-2697 v3 CPU (it can deliver 0.7 TFLOPS) is around $2700. For this case, GPUs ($575 per TFLOPS) are around 7-fold cheaper than CPUs ($3857 per TFLOPS).

Moreover, GPUs are inherently more energy efficient than other ways of computation, e.g., a CPU-only machine. This is because they are optimized for throughput and performance per Watt and not absolute performance. Taking Tianhe-1A as an example, which is capable of a maximum of 2.5 PFLOPS and was the fastest supercomputer in the world from October 2010 to June 2011, but it consumes only 4 megawatts, requiring 3-fold less power than a CPU-only machine. The power saved by using GPUs is enough to power a house for 8000 years [136].

6.5.2 Limitations

Although modern GPUs have been successfully introduced into many scientific research areas, it is still challenging to fully exploit their tremendous computational power. First of all, the theoretical speedup cannot be achieved for all algorithms, and in fact most of the algorithms cannot be fully parallelized. Gaining maximum speedup requires that GPUs have to be appropriately utilised and algorithms are coded to reflect the GPU architecture. So, it is necessary to understand the architecture and programming model of GPUs. GPU programming, which requires much more consideration about how to efficiently manage resources (e.g., parallel cores, memory), differs significantly from traditional CPUs. In particular, it is more difficult to incorporate GPU acceleration into existing codes. Moreover, since GPUs follow the SIMT execution model for parallel computing, the overall efficiency will be limited by any thread divergence within a warp. In order to achieve higher computation performance, it is necessary to carefully design the GPU program and limit thread divergence, e.g., the use of conditional and flow control statements should be avoided as much as possible. Moreover, GPUs have high memory latency, which may significantly affect performance. Particularly, global memory has the highest latency (400-800 cycles, around 300ns). To be more specific, input/output (I/O)

often dominates computation runtime, and global memory I/O is the slowest form on GPUs. Because of this particular reason, it is critical to limit global memory access to as little as possible. For GPU acceleration, it is inevitable that FLIM histogram data has to be transferred to global memory before FLIM analysis, and lifetime results have to be sent back to host memory. As mentioned in section 3.7.2, some programming techniques such as concurrent executions and coalesced memory accesses can be applied to minimize the global memory latency.

6.5.3 Further considerations

In order to fully exploit the power of GPUs, it is necessary to consider the histogram data format and FLIM algorithm development by understanding the architecture and programming model of GPUs. First, to avoid as much warp divergence as described in chapter 3, when generating FLIM histograms, the number of effective time bins is suggested to be a multiple of 32. This configuration also applies to FLIM algorithm development, e.g., when developing ANN-FLIM algorithms as demonstrated in section 6.4.2. Moreover, as we know that FLIM histograms are all integers (limited numbers of integers), so it is beneficial to build a lookup table instead of complicated calculation on GPUs. For example, for IRF based ANN-FLIM algorithms, the logarithmic scaling can be replaced by a pre-calculated lookup table, during pre-processing. In addition, CUDA profiler is able to help developers to further exam the program and identify the hotspots where the program optimization should be focused on.

6.6 Conclusions

A high-speed FLIM system is a powerful system to measure dynamic protein-protein interactions in live cells. It can be used for a wide variety of applications, e.g., cancer diagnosis, brain tumor surgery, drug development. However, FLIM systems require significant data acquisition and data analysis. Recent development of CMOS technologies has significantly improved FLIM data acquisition with high resolution and incredible speed. This dramatic improvement, however, produces a huge amount of data, and makes

the data analysis more and more challenging. In terms of real-time FLIM technologies, FLIM data analysis has really become a bottleneck. To solve this challenging problem and build a high-speed FLIM analysis system, this research was dedicated to harvesting the power of high-speed parallel processors – general purpose GPUs. In addition to accelerating standard algorithms, this research also introduced new high-speed algorithms for FLIM analysis.

First, some background information and the principle of FLIM have been described, to demonstrate why it is important to develop FLIM techniques and how to use this technology. Also, standard FLIM analysis algorithms have been demonstrated, before they were implemented in GPU computing.

GPUs have become more and more popular in a wide range of research areas, and they can also be used for FLIM analysis. By exploring the architecture of GPUs and CUDA programming techniques, this research gives a general understanding about how GPUs work and how to implement FLIM algorithms on GPUs.

In addition, a basic knowledge of ANNs has been demonstrated, as well as the potential for FLIM analysis. It has been proved that a well-trained feedforward neural network is able to perform FLIM analysis. Besides, without any iteration operations, ANNs are capable of delivering high-speed FLIM analysis.

In conclusion, a GPU accelerated high-speed FLIM analysis system has been introduced, which contains both standard algorithms and ANN-FLIMs. The potential of this system has been evaluated based on synthesized data and experimental data. This system provides a comprehensive FLIM analysis solution with much higher speed than CPU-only systems, which allows users to choose appropriate algorithms with certain requirements for different FLIM applications. Moreover, real-time FLIM analysis can be achieved with high-speed algorithms (i.e., simple algorithms and ANN-FLIMs).

6.7 Recommendations for future research

For the past decade, general purpose GPUs have evolved dramatically, and GPU accelerators are now powering hundreds of different areas, including businesses, universities, and governments. More and more advanced technologies are being introduced in GPU computing, from which FLIM analysis can obtain further benefits and additional speedup can be achieved. For example, dynamic parallelism, which is an extension to the CUDA programming model, enables a CUDA kernel to create and synchronize the additional work directly on the GPU. This feature allows launching CUDA kernels directly from other kernels, and enables further speedups by avoiding CPU/GPU interactions. Moreover, multi-GPU technology is a combination of multiple professional GPUs to increase computing horsepower. This technology allows GPUs to intelligently scale the performance of the application and dramatically speed up computation by wisely allocating GPU resources to applications as needed [137]. This is of significance to FLIM analysis, when the data is growing dramatically. Also, multi-GPU technology is able to massively speedup and improve the ANN training process.

In addition to optimize the ANN-FLIM methods, it is also worth considering other ML algorithms. With the increasing amounts of FLIM data, the need for advanced and automated methods for FLIM analysis grows. As a subfield of computer science, ML allows computers to learn without being explicitly programmed [92, 138]. ML algorithms can first automatically detect patterns in the given data, then apply learned patterns to future data and predict outcomes of interest [105]. For FLIM analysis, ML algorithms with supervised learning are of interest.

References

[1] F. J *et al.*, "GLOBOCAN 2012 v1.0, Cancer Incidence and Mortality Worldwide: IARC CancerBase No. 11 [Internet]," http://globocan.iarc.fr.

[2] M. Ingaramo *et al.*, "Two-photon-like microscopy with orders-of-magnitude lower illumination intensity via two-step fluorescence," *Nat Commun,* vol. 6, pp. 8184, 2015.

[3] L. Gu *et al.*, "In vivo time-gated fluorescence imaging with biodegradable luminescent porous silicon nanoparticles," *Nat Commun,* vol. 4, pp. 2326, 2013.

[4] J. Goedhart *et al.*, "Bright cyan fluorescent protein variants identified by fluorescence lifetime screening," *Nat Methods,* vol. 7, no. 2, pp. 137-9, Feb, 2010.

[5] L. C. Chen *et al.*, "Fluorescence Lifetime Imaging Microscopy for Quantitative Biological Imaging," *Digital Microscopy, 4th Edition,* vol. 114, pp. 457-488, 2013.

[6] J. R. Lakowicz, *Principles of fluorescence spectroscopy*, 3rd ed., New York: Springer, 2006.

[7] W. Becker, "Fluorescence lifetime imaging--techniques and applications," *J Microsc,* vol. 247, no. 2, pp. 119-36, Aug, 2012.

[8] T. Forster, "*Zwischenmolekulare Energiewanderung Und Fluoreszenz," *Annalen Der Physik,* vol. 2, no. 1-2, pp. 55-75, 1948.

[9] A. R. Clapp *et al.*, "Forster resonance energy transfer investigations using quantum-dot fluorophores," *Chemphyschem,* vol. 7, no. 1, pp. 47-57, Jan 16, 2006.

[10] R. M. Clegg, "Fluorescence resonance energy transfer," *Current opinion in biotechnology,* vol. 6, no. 1, pp. 103-110, 1995.

[11] J. McGinty *et al.*, "Wide-field fluorescence lifetime imaging of cancer," *Biomed Opt Express,* vol. 1, no. 2, pp. 627-640, 2010.

[12] Y. Sun *et al.*, "Fluorescence lifetime imaging microscopy for brain tumor image-guided surgery," *J Biomed Opt,* vol. 15, no. 5, pp. 056022, Sep-Oct, 2010.

[13] K. Okabe *et al.*, "Intracellular temperature mapping with a fluorescent polymeric thermometer and fluorescence lifetime imaging microscopy," *Nature Communications,* vol. 3, Feb, 2012.

[14] M. Nobis *et al.*, "Intravital FLIM-FRET imaging reveals dasatinib-induced spatial control of src in pancreatic cancer," *Cancer Research,* vol. 73, no. 15, pp. 4674-86, Aug 1, 2013.

[15] C. De Los Santos *et al.*, "FRAP, FLIM, and FRET: Detection and analysis of cellular dynamics on a molecular scale using fluorescence microscopy," *Molecular Reproduction and Development,* vol. 82, no. 7-8, pp. 587-604, 2015.

[16] T. Omer *et al.*, "Reduced temporal sampling effect on accuracy of time-domain fluorescence lifetime Forster resonance energy transfer," *J Biomed Opt,* vol. 19, no. 8, pp. 086023, Aug, 2014.

[17] J. Sytsma *et al.*, "Time-gated fluorescence lifetime imaging and microvolume spectroscopy using two-photon excitation," *Journal of Microscopy-Oxford,* vol. 191, pp. 39-51, Jul, 1998.

[18] D. M. Grant *et al.*, "High speed optically sectioned fluorescence lifetime imaging permits study of live cell signaling events," *Optics express,* vol. 15, no. 24, pp. 15656-15673, 2007.

[19] W. Becker, *Advanced time-correlated single photon counting techniques*, Berlin ; New York: Springer, 2005.

[20] S. P. Poland *et al.*, "A high speed multifocal multiphoton fluorescence lifetime imaging microscope for live-cell FRET imaging," *Biomedical Optics Express,* vol. 6, no. 2, pp. 277-96, Feb 1, 2015.

[21] L. Turgeman *et al.*, "Photon efficiency optimization in time-correlated single photon counting technique for fluorescence lifetime imaging systems," *IEEE Trans Biomed Eng,* vol. 60, no. 6, pp. 1571-9, Jun, 2013.

[22] S. P. Poland *et al.*, "A high speed multifocal multiphoton fluorescence lifetime imaging microscope for live-cell FRET imaging," *Biomedical Optics Express,* vol. 6, no. 2, pp. 277-296, 2015.

[23] X. Michalet *et al.*, "Development of new photon-counting detectors for single-molecule fluorescence microscopy," *Philos Trans R Soc Lond B Biol Sci,* vol. 368, no. 1611, pp. 20120035, Feb 5, 2013.

[24] E. Charbon, "Single-photon imaging in complementary metal oxide semiconductor processes," *Philos Trans A Math Phys Eng Sci,* vol. 372, no. 2012, pp. 20130100, Mar 28, 2014.

[25] R. M. Field *et al.*, "A 100 fps, time-correlated single-photon-counting-based fluorescence-lifetime imager in 130 nm CMOS," *Solid-State Circuits, IEEE Journal of,* vol. 49, no. 4, pp. 867-880, 2014.

[26] S. J. Herschel, "On a case of superficial colour presented by a homogeneous liquid internally colourless," *Philosophical Transactions of the Royal Society of London,* vol. 135, pp. 143-145, 1845.

[27] A. Jablonski, " Über den mechanisms des photolumineszenz von farbstoffphosphores," *Z phys,* vol. 94, pp. 38-46, 1935.

[28] G. G. Stokes, "On the change of refrangibility of light," *Philosophical Transactions of the Royal Society of London,* vol. 142, pp. 463-562, 1852.

[29] D. U. Li *et al.*, "Hardware implementation algorithm and error analysis of high-speed fluorescence lifetime sensing systems using center-of-mass method," *Journal of Biomedical Optics,* vol. 15, no. 1, Jan-Feb, 2010.

[30] R. R. Gaddam *et al.*, "Fluorescence spectroscopy of nanofillers and their polymer nanocomposites," *Spectroscopy of Polymer Nanocomposites*, pp. 158, 2016.

[31] C. W. Chang *et al.*, "Fluorescence lifetime imaging microscopy," *Digital Microscopy, 3rd Edition,* vol. 81, pp. 495-+, 2007.

[32] E. Gratton *et al.*, "Fluorescence lifetime imaging for the two-photon microscope: time-domain and frequency-domain methods," *J Biomed Opt,* vol. 8, no. 3, pp. 381-90, Jul, 2003.

[33] S. Kumar *et al.*, "Multifocal multiphoton excitation and time correlated single photon counting detection for 3-D fluorescence lifetime imaging," *Opt Express,* vol. 15, no. 20, pp. 12548-61, Oct 1, 2007.

[34] J. L. Rinnenthal *et al.*, "Parallelized TCSPC for dynamic intravital fluorescence lifetime imaging: quantifying neuronal dysfunction in neuroinflammation," *PLoS One,* vol. 8, no. 4, pp. e60100, 2013.

[35] D. D. U. Li *et al.*, "Video-rate fluorescence lifetime imaging camera with CMOS single-photon avalanche diode arrays and high-speed imaging algorithm," *Journal of Biomedical Optics,* vol. 16, no. 9, Sep, 2011.

[36] W. Becker, "Recording the instrument response function of a multiphoton FLIM system," *Application Notes, www. becker-hickl. com*, 2007.

[37] P. Hall *et al.*, "Better Estimates of Exponential Decay Parameters," *Journal of Physical Chemistry,* vol. 85, no. 20, pp. 2941-2946, 1981.

[38] A. A. Istratov *et al.*, "Exponential analysis in physical phenomena," *Review of Scientific Instruments,* vol. 70, no. 2, pp. 1233-1257, Feb, 1999.

[39] P. R. Barber *et al.*, "Global and pixel kinetic data analysis for FRET detection by multi-photon time-domain FLIM," *Multiphoton Microscopy in the Biomedical Sciences V,* vol. 5700, pp. 171-181, 2005.

[40] S. C. Warren *et al.*, "Rapid Global Fitting of Large Fluorescence Lifetime Imaging Microscopy Datasets," *Plos One,* vol. 8, no. 8, Aug 5, 2013.

[41] P. J. Verveer *et al.*, "Global analysis of fluorescence lifetime imaging microscopy data," *Biophys J,* vol. 78, no. 4, pp. 2127-37, Apr, 2000.

[42] D. U. Li *et al.*, "Real-time fluorescence lifetime imaging system with a 32 x 32 0.13microm CMOS low dark-count single-photon avalanche diode array," *Opt Express,* vol. 18, no. 10, pp. 10257-69, May 10, 2010.

[43] Y. J. Won *et al.*, "Precision and accuracy of the analog mean-delay method for high-speed fluorescence lifetime measurement," *J Opt Soc Am A Opt Image Sci Vis,* vol. 28, no. 10, pp. 2026-32, Oct 1, 2011.

[44] A. Leray *et al.*, "Spatio-temporal quantification of FRET in living cells by fast time-domain FLIM: a comparative study of non-fitting methods," *PLoS One,* vol. 8, no. 7, pp. e69335, 2013.

[45] D. D.-U. Li *et al.*, “Fast bi-exponential fluorescence lifetime imaging analysis methods,” *Optics Letters,* vol. 40, no. 3, pp. 336-339, 2015.

[46] W. R. Ware *et al.*, “Deconvolution of Fluorescence and Phosphorescence Decay Curves - Least-Squares Method,” *Journal of Physical Chemistry,* vol. 77, no. 17, pp. 2038-2048, 1973.

[47] J. Demas, *Excited state lifetime measurements*: Elsevier, 2012.

[48] I. Isenberg, “Robust Estimation in Pulse Fluorometry - a Study of the Method of Moments and Least-Squares,” *Biophysical Journal,* vol. 43, no. 2, pp. 141-148, 1983.

[49] J. R. Lakowicz, *Principles of fluorescence spectroscopy*: Springer Science & Business Media, 2013.

[50] J. A. Jo *et al.*, “Ultrafast method for the analysis of fluorescence lifetime imaging microscopy data based on the Laguerre expansion technique,” *Ieee Journal of Selected Topics in Quantum Electronics,* vol. 11, no. 4, pp. 835-845, Jul-Aug, 2005.

[51] Y. Zhang *et al.*, “Optimizing Laguerre expansion based deconvolution methods for analysing bi-exponential fluorescence lifetime images,” *Optics Express,* vol. 24, no. 13, pp. 13894-13905, 2016.

[52] M. I. Rowley *et al.*, “Robust Bayesian Fluorescence Lifetime Estimation, Decay Model Selection and Instrument Response Determination for Low-Intensity FLIM Imaging,” *PLoS One,* vol. 11, no. 6, pp. e0158404, 2016.

[53] M. A. Digman *et al.*, “The phasor approach to fluorescence lifetime imaging analysis,” *Biophysical Journal,* vol. 94, no. 2, pp. L14-L16, Jan 15, 2008.

[54] J. J. Moré, "The Levenberg-Marquardt algorithm: implementation and theory," *Numerical analysis*, pp. 105-116: Springer, 1978.

[55] J. M. I. Maarek *et al.*, “Time-resolved fluorescence spectra of arterial fluorescent compounds: Reconstruction with the laguerre expansion technique,” *Photochemistry and Photobiology,* vol. 71, no. 2, pp. 178-187, Feb, 2000.

[56] J. Liu *et al.*, “A novel method for fast and robust estimation of fluorescence decay dynamics using constrained least-squares deconvolution with Laguerre expansion,” *Physics in Medicine and Biology,* vol. 57, no. 4, pp. 843-865, Feb 21, 2012.

[57] K. Suhling *et al.*, “Fluorescence lifetime imaging (FLIM): Basic concepts and some recent developments,” *Medical Photonics,* vol. 27, pp. 3-40, 2015.

[58] H. Wallrabe *et al.*, “Imaging protein molecules using FRET and FLIM microscopy,” *Current Opinion in Biotechnology,* vol. 16, no. 1, pp. 19-27, Feb, 2005.

[59] K. V. Kuchibhotla *et al.*, “Synchronous hyperactivity and intercellular calcium waves in astrocytes in Alzheimer mice,” *Science,* vol. 323, no. 5918, pp. 1211-5, Feb 27, 2009.

[60] C. Hille *et al.*, “Time-domain fluorescence lifetime imaging for intracellular pH sensing in living tissues,” *Analytical and Bioanalytical Chemistry,* vol. 391, no. 5, pp. 1871-1879, Jul, 2008.

[61] W. Becker, “Fluorescence lifetime imaging - techniques and applications,” *Journal of Microscopy,* vol. 247, no. 2, pp. 119-136, Aug, 2012.

[62] F. K. Zhou *et al.*, “Molecular Rotors as Fluorescent Viscosity Sensors: Molecular Design, Polarity Sensitivity, Dipole Moments Changes, Screening Solvents, and Deactivation Channel of the Excited States,” *European Journal of Organic Chemistry*, no. 25, pp. 4773-4787, Sep, 2011.

[63] M. K. Kuimova *et al.*, “Molecular rotor measures viscosity of live cells via fluorescence lifetime imaging,” *Journal of the American Chemical Society,* vol. 130, no. 21, pp. 6672-+, May 28, 2008.

[64] J. A. Levitt *et al.*, “Membrane-Bound Molecular Rotors Measure Viscosity in Live Cells via Fluorescence Lifetime Imaging,” *Journal of Physical Chemistry C,* vol. 113, no. 27, pp. 11634-11642, Jul 9, 2009.

[65] R. M. Field *et al.*, “A 100 fps, time-correlated single-photon-counting-based fluorescence-lifetime imager in 130 nm CMOS,” *IEEE Journal of Solid-State Circuits,* vol. 49, no. 4, pp. 867-880, 2014.

[66] R. Buyya *et al.*, *Mastering cloud computing: foundations and applications programming*: Newnes, 2013.

[67] D. Kirk *et al.*, *Programming massively parallel processors hands-on with CUDA*, Burlington, MA: Morgan Kaufmann Publishers, 2010.

[68] N. Corporation, "CUDA C Programming guide," 2016.

[69] C. Woolley, "CUDA Overview," http://www.cc.gatech.edu/~vetter/keeneland/tutorial-2012-02-20/06-cuda-overview.pdf, 21/11/16.

[70] P. N. Glaskowsky, “NVIDIA’s Fermi: the first complete GPU computing architecture,” *White paper,* vol. 18, 2009.

[71] N. Corporation, “Fermi Compute Architecture Whitepaper,” 2009.

[72] NVIDIA, "NVIDIA Launches the World's First Graphics Processing Unit: GeForce 256," http://www.nvidia.com/object/IO_20020111_5424.html.

[73] S. Jamshed, *Using HPC for Computational Fluid Dynamics: A Guide to High Performance Computing for CFD Engineers*: Academic Press, 2015.

[74] N. Corporation, "NVIDIA CUDA Architecture Introduction & Overview," 2009.

[75] J. Sanders *et al.*, *CUDA by example : an introduction to general-purpose GPU programming*, Upper Saddle River ; London: Addison-Wesley, 2011.

[76] J. Cheng *et al.*, *Professional Cuda C Programming*: John Wiley & Sons, 2014.

[77] "GPU-ACCELERATED APPLICATIONS," http://www.nvidia.com/content/gpu-applications/PDF/gpu-applications-catalog.pdf, 2016 Nov 30.

[78] M. Giles *et al.*, "GPU implementation of finite difference solvers." pp. 1-8.

[79] J. Michalakes *et al.*, “GPU acceleration of numerical weather prediction,” *2008 Ieee International Symposium on Parallel & Distributed Processing, Vols 1-8*, pp. 2219-2225, 2008.

[80] D. Singh *et al.*, “A survey on platforms for big data analytics,” *Journal of Big Data,* vol. 2, no. 1, pp. 1, 2014.

[81] Y. Jia *et al.*, "Caffe: Convolutional architecture for fast feature embedding." pp. 675-678.

[82] M. Smelyanskiy *et al.*, “Mapping High-Fidelity Volume Rendering for Medical Imaging to CPU, GPU and Many-Core Architectures,” *Ieee Transactions on Visualization and Computer Graphics,* vol. 15, no. 6, pp. 1563-1570, Nov-Dec, 2009.

[83] K.-C. Lee *et al.*, "System and method for site abnormality recording and notification," Google Patents, 2013.

[84] J. Nickolls *et al.*, “The Gpu Computing Era,” *Ieee Micro,* vol. 30, no. 2, pp. 56-69, Mar-Apr, 2010.

[85] G. Wu *et al.*, “GPU acceleration of time-domain fluorescence lifetime imaging,” *J Biomed Opt,* vol. 21, no. 1, pp. 17001, Jan, 2016.

[86] N. Corporation, "CUDA C Programming guide," 2014.

[87] J. A. Goodman *et al.*, “Accelerating an Imaging Spectroscopy Algorithm for Submerged Marine Environments Using Graphics Processing Units,” *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing,* vol. 4, no. 3, pp. 669-676, Sep, 2011.

[88] M. Harris, “Optimizing parallel reduction in CUDA,” *NVIDIA Developer Technology,* vol. 2, no. 4, 2007.

[89] N. Wilt, *The CUDA handbook : a comprehensive guide to GPU programming*, Boston, Mass. ; London: Addison-Wesley, 2013.

[90] NVIDIA, “CUDA C Best Practices Guide,” *NVIDIA Corporation*, 2014.

[91] J. McGinty *et al.*, “Multidimensional fluorescence imaging,” *Fret and Flim Techniques,* vol. 33, pp. 133-169, 2009.

[92] P. Domingos, “A Few Useful Things to Know About Machine Learning,” *Communications of the Acm,* vol. 55, no. 10, pp. 78-87, Oct, 2012.

[93] S. S. Haykin *et al.*, *Neural networks and learning machines*, 3rd ed., New York: Prentice Hall, 2009.

[94] J. Schmidhuber, “Deep learning in neural networks: An overview,” *Neural Networks,* vol. 61, pp. 85-117, Jan, 2015.

[95] S. Sumathi *et al.*, *Solar PV and Wind Energy Conversion Systems*: Springer International Publishing, 2015.

[96] D. Anderson *et al.*, “Artificial neural networks technology,” *Kaman Sciences Corporation,* vol. 258, no. 6, pp. 1-83, 1992.

[97] G. Paths, “Design and analysis of algorithms,” 1974.

[98] W. S. Mcculloch *et al.*, “A Logical Calculus of the Ideas Immanent in Nervous Activity,” *Bulletin of Mathematical Biology,* vol. 52, no. 1-2, pp. 99-115, 1943.

[99] “The organization of behavior,” *Annee Psychologique,* vol. 51, pp. 493-494, 1949.

[100] P. Werbos, “Beyond regression: New tools for prediction and analysis in the behavioral sciences,” 1974.

[101] M. T. Hagan *et al.*, *Neural Network Design (2nd Edition)*: Martin Hagan, 2014.

[102] V. Nair *et al.*, "Rectified linear units improve restricted boltzmann machines." pp. 807-814.

[103] K. Hornik *et al.*, “Multilayer Feedforward Networks Are Universal Approximators,” *Neural Networks,* vol. 2, no. 5, pp. 359-366, 1989.

[104] R. E. Schapire *et al.*, “Foundations of Machine Learning,” *Boosting: Foundations and Algorithms*, pp. 23-52, 2012.

[105] K. P. Murphy, “Machine Learning: A Probabilistic Perspective,” *Machine Learning: A Probabilistic Perspective*, pp. 1-1067, 2012.

[106] P. L. Bartlett, “An introduction to reinforcement learning theory: Value function methods,” *Advanced Lectures on Machine Learning,* vol. 2600, pp. 184-202, 2002.

[107] D. E. Rumelhart *et al.*, *Learning internal representations by error propagation*, DTIC Document, 1985.

[108] R. Taormina *et al.*, “Artificial neural network simulation of hourly groundwater levels in a coastal aquifer system of the Venice lagoon,” *Engineering Applications of Artificial Intelligence,* vol. 25, no. 8, pp. 1670-1676, Dec, 2012.

[109] F. Hussain *et al.*, “Efficient Deep Neural Network for Digital Image Compression Employing Rectified Linear Neurons,” *Journal of Sensors*, 2016.

[110] S. Lee *et al.*, “A multi-industry bankruptcy prediction model using back-propagation neural network and multivariate discriminant analysis,” *Expert Systems with Applications,* vol. 40, no. 8, pp. 2941-2946, Jun 15, 2013.

[111] S. Poungponsri *et al.*, “An adaptive filtering approach for electrocardiogram (ECG) signal noise reduction using neural networks,” *Neurocomputing,* vol. 117, pp. 206-213, Oct 6, 2013.

[112] G. Hinton *et al.*, “Deep Neural Networks for Acoustic Modeling in Speech Recognition,” *Ieee Signal Processing Magazine,* vol. 29, no. 6, pp. 82-97, Nov, 2012.

[113] L. Chen *et al.*, "Beyond human recognition: A CNN-based framework for handwritten character recognition." pp. 695-699.

[114] K. A. Walther *et al.*, “Precise measurement of protein interacting fractions with fluorescence lifetime imaging microscopy,” *Mol Biosyst,* vol. 7, no. 2, pp. 322-36, Feb, 2011.

[115] J. Jiang *et al.*, "Medical image analysis with artificial neural networks," *Comput Med Imaging Graph,* vol. 34, no. 8, pp. 617-31, Dec, 2010.

[116] G. Wu *et al.*, "Artificial neural network approaches for fluorescence lifetime imaging techniques," *Optics Letters,* vol. 41, no. 11, pp. 2561-2564, Jun 1, 2016.

[117] J. M. Hammersley *et al.*, *Monte Carlo methods*, London: Wiley, 1964.

[118] L. Kocis *et al.*, "Computational investigations of low-discrepancy sequences," *Acm Transactions on Mathematical Software,* vol. 23, no. 2, pp. 266-294, Jun, 1997.

[119] J. Enderlein *et al.*, "A maximum likelihood estimator to distinguish single molecules by their fluorescence decays," *Chemical Physics Letters,* vol. 270, no. 5-6, pp. 464-470, May 30, 1997.

[120] T. A. Laurence *et al.*, "Efficient maximum likelihood estimator fitting of histograms," *Nat Methods,* vol. 7, no. 5, pp. 338-9, May, 2010.

[121] H. J. Kim *et al.*, "Hybrid neural network approach in description and prediction of dynamic behavior of chaotic chemical reaction systems," *Korean Journal of Chemical Engineering,* vol. 17, no. 6, pp. 696-703, Nov, 2000.

[122] M. T. Hagan *et al.*, *Neural network design*, 1st ed., Boston: PWS Pub., 1996.

[123] K. S. Oh *et al.*, "GPU implementation of neural networks," *Pattern Recognition,* vol. 37, no. 6, pp. 1311-1314, Jun, 2004.

[124] J. M. Nageswaran *et al.*, "Efficient Simulation of Large-Scale Spiking Neural Networks Using CUDA Graphics Processors," *Proceedings of Intl. Joint Conf. on Neural Networks,* , pp. 2145-2152, 2009.

[125] T. F. Coleman *et al.*, "An interior trust region approach for nonlinear minimization subject to bounds," *Siam Journal on Optimization,* vol. 6, no. 2, pp. 418-445, May, 1996.

[126] H. C. Gerritsen *et al.*, "Fluorescence lifetime imaging in scanning microscopes: acquisition speed, photon economy and lifetime resolution," *Journal of Microscopy-Oxford,* vol. 206, pp. 218-224, Jun, 2002.

[127] A. Esposito *et al.*, "Optimizing frequency-domain fluorescence lifetime sensing for high-throughput applications: photon economy and acquisition speed," *Journal of the Optical Society of America a-Optics Image Science and Vision,* vol. 24, no. 10, pp. 3261-3273, Oct, 2007.

[128] H. E. Grecco *et al.*, "Global analysis of time correlated single photon counting FRET-FLIM data," *Opt Express,* vol. 17, no. 8, pp. 6493-508, Apr 13, 2009.

[129] M. Lahn *et al.*, "Two-photon microscopy and fluorescence lifetime imaging reveal stimulus-induced intracellular Na+ and Cl- changes in cockroach salivary acinar cells," *Am J Physiol Cell Physiol,* vol. 300, no. 6, pp. C1323-36, Jun, 2011.

[130] D. Wustner *et al.*, "Quantitative assessment of sterol traffic in living cells by dual labeling with dehydroergosterol and BODIPY-cholesterol," *Chem Phys Lipids,* vol. 164, no. 3, pp. 221-35, Mar, 2011.

[131] R. Cundall, *Time-resolved fluorescence spectroscopy in biochemistry and biology*: Springer Science & Business Media, 2013.

[132] A. Benda *et al.*, “Fluorescence lifetime correlation spectroscopy combined with lifetime tuning: new perspectives in supported phospholipid bilayer research,” *Langmuir,* vol. 22, no. 23, pp. 9580-5, Nov 07, 2006.

[133] M. Stockl *et al.*, “Detection of lipid domains in model and cell membranes by fluorescence lifetime imaging microscopy of fluorescent lipid analogues,” *J Biol Chem,* vol. 283, no. 45, pp. 30828-37, Nov 07, 2008.

[134] S. P. Poland *et al.*, “New high-speed centre of mass method incorporating background subtraction for accurate determination of fluorescence lifetime,” *Optics Express,* vol. 24, no. 7, pp. 6899-6915, 2016/04/04, 2016.

[135] S. Chakraborty *et al.*, “Quantification of the Metabolic State in Cell-Model of Parkinson's Disease by Fluorescence Lifetime Imaging Microscopy,” *Sci Rep,* vol. 6, pp. 19145, 2016.

[136] NVIDIA, "Energy Efficiency," http://www.nvidia.co.uk/object/gcr-energy-efficiency.html, 20 March 2017.

[137] NVIDIA, "NVIDIA Multi-GPU Technology," http://www.nvidia.com/object/multi-gpu-technology.html, 20 March 2017.

[138] A. L. Samuel, “Some Studies in Machine Learning Using the Game of Checkers,” *Ibm Journal of Research and Development,* vol. 3, no. 3, pp. 211-&, 1959.